Final Report For ONR Contract N00014-86-K-0167

PI's: Geoffrey Hinton and James L. McClelland
Departments of Computer Science and Psychology
Carnegie Mellon University

20 April 1990

During the course of the above-referenced contract several research projects were carried out. These included:

1. The Development of the Time-Delay Neural Network Architecture for Speech Recognition (Lang and Hinton, 1988).
2. Learning Representations by Recirculation. (Hinton and McClelland, 1988).
3. Applying Contextual Constraints in Sentence Comprehension. (St. John and McClelland, 1988).
4. Using Fast Weights to Deblur Old Memories. (Hinton and Plaut, 1987).

Papers describing each of these developments are appended as elements of this report.

References

Lang, K.J. & Hinton, G.E. (1988). The development of the time-delay neural network architecture for speech recognition. Technical Report CMU-CS-11-152, Department of Computer Science, Carnegie Mellon University.

Hinton, G.E., & McClelland, J.L. (1988). Learning representations by recirculation. In D.Z. Anderson (Ed.), *Neural information processing systems*, New York: American Institute of Physics.

St. John, M.F. & McClelland, J.L. (1988). Applying contextual constraints in sentence comprehension. Proceedings of the Cognitive Science Society.

Hinton, G.E. & Plaut, D.C. (1987). Using fast weights to deblur old memories. Proceedings of the Cognitive Science Society.

The Development of the Time-Delay Neural Network Architecture for Speech Recognition

Tech Report CMU-CS-88-152

Kevin J. Lang
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Geoffrey E. Hinton
Computer Science and Psychology Departments
University of Toronto
10 Kings College Road
Toronto, Ontario M5S 1A4
Canada

This work was supported by grants from ARPA, ONR, and the System Development Foundation. Geoffrey Hinton is a fellow of the Canadian Institute for Advanced Research.

1. Introduction

Currently, one of the most powerful connectionist learning procedures is back-propagation [Rumelhart 86], which repeatedly adjusts the weights in a network so as to minimize a measure of the difference between the actual output vector of the network and a desired output vector given the current input vector. The simple weight adjusting rule is derived by propagating partial derivatives of the error backwards through the net using the chain rule. Experiments have shown that back-propagation has most of the properties desired by connectionists. As with any worthwhile learning rule, it can learn non-linear black box functions and make fine distinctions between input patterns in the presence of noise. Moreover, starting from random initial states, back-propagation networks can learn to use their hidden (intermediate layer) units to efficiently represent the structure that is inherent in their input data, often discovering intuitively pleasing features.

The fact that back-propagation can discover features and distinguish between similar patterns in the presence of noise makes it a natural candidate as a speech recognition method. Another reason for expecting back-propagation to be be good at speech is the success that hidden Markov models have enjoyed in speech recognition; both approaches are based on the idea that a simplistic model of speech can be useful when there is a rigorous automatic method for tuning its parameters.

Our investigation of back-propagation on real data (as opposed to synthetic problems such as the one described in [Plaut 86]) began with a set of recordings of the isolated words "bee", "dee", "ee", and "vee" that we obtained from Peter Brown. We chose this domain because it was difficult enough to be interesting but small enough to be computationally feasable for us, and because we could compare our performance with an ironclad benchmark, namely that of the enhanced Hidden Markov model that Brown had developed on this data set for the purposes of his thesis [Brown 87]. The task is difficult because the four letters are the most confusable subset of the alphabet and because the recordings were made in an office environment with a remote microphone. Human performance on the task is approximately 95%, while the standard IBM 20,000 word HMM can only recognize 80%. Brown's best HMM was able to classify 89% of the test set correctly.

After experimenting with a variety of signal processing schemes and networks, we found that good results could be obtained by feeding spectrograms into a new type of network whose architecture allowed it to locate and analyze the most informative region of a spectrogram regardless of its time alignment. When this network was trained using a multi-resolution training technique inspired by vision research, it was able to generalize to 94% of the testing corpus.

2. Getting Started

2.1 The task

The data set for our experiments was originally recorded by the speech group at IBM and was used by Peter Brown as the domain for his thesis research. There are approximately 800 utterances in all, generated by 100 male speakers who said each of the four words in the set {"bee", "dee", "ee", "vee"}[1] twice, once for training and once for testing. The microphone was sitting on a disk drive in an office, resulting in a signal-to-noise ratio of about 10dB at best, depending on the word. The researchers at IBM threw away a number of obviously erroneous cases, leaving a total of 372 training and 396 test cases. The task isn't truly speaker independent because the same 100 speakers are used both for training and for testing. In order to reduce the quantity of data that he needed to send us, Brown performed a Viterbi alignment that located the consonant-vowel transition point of each utterance. This made it possible to clip out 144 milliseconds around the transition, discarding the steady-state portions of the word.

We downsampled these recordings from 20 to 16 thousand samples per second, and then used a standard DFT program to generate spectrograms containing 128 frequency bands ranging up to 8 kHz and 48 time frames of 3 msec each. Thus each of the cases was turned into a spectrogram containing 6144 energy values. Background noise was clearly visible on the spectrograms, as were occasional but severe time alignment problems.

2.2 The network model

Back-propagation networks compute real-valued vector functions. We structured our networks to map spectrograms to 4-tuples that represent probabilities for each of the four words **B**, **D**, **E**, and **V**. When a spectrogram is presented on the input of such a network, activation flows up through the connections, and each output unit turns on by an amount that indicates its confidence that the spectrogram is an instance of its own word. For training purposes, we specify that the activation of the correct output unit should be at least 0.8 and that the activation of the other three output units should be at most 0.2. For testing purposes, the most activated unit is considered to be the network's answer (this is the "best guess" rule.)

The networks described in section 2 are all of the simple 2-layer type, in which the input layer is directly connected to the output layer without any hidden units in between. A schematic diagram of a 2-layer network is shown in figure 2-5, along with weights (connection strengths) selected by the learning procedure to enable the network to perform a sensible mapping from spectrograms into the 4-dimensional word space.

[1]These words will henceforth be denoted by **B**, **D**, **E**, and **V**.

2.3 The need for spectrogram post-processing

As mentioned above, the spectrograms that we intended to use as the input to our networks contained 48×128=6144 points. A back-propagation network must have at least one connection to each of its input units, and every connection contains a weight which must be trained.[2]. Clearly, 372 training examples are insufficient to train a model with some multiple of 6144 parameters. Since the size of the training corpus couldn't be increased, we needed to decrease the size of our model, and hence the size of the input spectrograms. This was accomplished by combining adjacent columns and rows of the raw spectrograms to generate smaller, less detailed spectrograms to feed into the network. It was also important to scale the energy values to the range that back-progation networks find palatable, namely between 0.0 and 1.0 without too many values near 0.5.

There are many plausible sounding methods of compressing a spectrogram and normalizing its component values. To avoid selecting a method arbitrarily, we set up a space of alternatives and then measured the performance that was possible using spectrograms generated by each method in the space. There were 4 variables in our space of spectrogram post-processing alternatives: the number of time steps in the spectrogram, the method for combining frequency bands, the method for normalizing energy values, and the choice of a shaping function to run the normalized values through before presenting them to the inputs of a network. The last three alternatives were explored in the experiments described in the next section, while the question of how much time resolution to include in the compressed spectrograms was deferred (see section 2.5).

2.4 Some alternatives

Following common speech recognition practice, we fixed the frequency resolution of our processed spectrograms at 16 bands. However, a choice needed to be made as to the method for condensing the 128 points contained in each time step of the raw spectrograms. The first method we considered, summing adjacent bands to create new ones, would preserve the linearity of the frequency scale. Alex Waibel suggested the second method: creating a mel scale with variable-width, overlapping bands. The mel scale was originally motivated by the properties of the cochlea, which has good frequency resolution at low frequencies and good temporal resolution at high frequencies. Unlike a cochlear model, fixed window DFT's can't make that tradeoff, but we still expected the mel scaled spectrograms to work better than the linear scaled ones because they provide proportionally more frequency resolution in the more informative lower frequency regions.

We evaluated two possibilities for an energy normalization method. The casewise method was to find the lowest and highest energies in a given spectrogram and then peg them at 0.0 and 1.0 respectively. The alternative was to globally choose energies to map to 0.0 and 1.0; after examining the overall distribution of energies in our data set, we selected the values of -5 and 105 decibels and then clipped any peaks that exceeded those bounds. We expected that the global method would yield better performance if the total

[2]In section 3.4 we will describe a constrained learning procedure that effectively reduces the number of free parameters in a network

energy in a spectrogram was an important clue to the identity of the word.

Finally, because the normalized spectrograms looked like they contained too many values around 0.5, we decided to transform the energy values with a shaping function that would drive them towards the boundary values of 0.0 and 1.0. The three alternatives we tested were doing nothing to the values, running them through a sigmoid, and squaring them (and then multiplying by 1.4). Figure 2-1 shows what a spectogram looks like in these three forms.

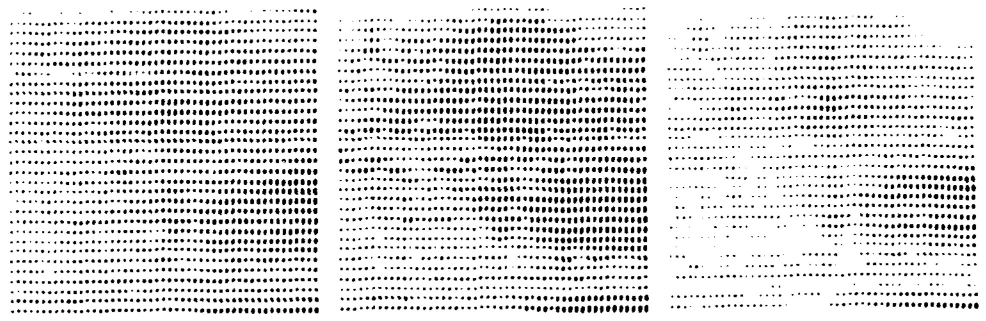


Figure 2-1:
A spectrogram for "vee" in raw, sigmoidized, and squared forms.

The generalization performance of a 2-layer network was measured on environments[3] generated using various combinations of the spectrogram post-processing rules described above. A 2-layer network is particularly useful in this sort of role not only because it converges rapidly, but also because its solutions are always the best possible for the given network with a particular environment; there is no chance of falling into a local minimum that would result in an unfair measure of the quality of the environment.[4]

Because we had no empirical evidence at this point regarding the best temporal resolution for a compressed spectrogram, we used a rough-and-ready argument along the lines of the one contained in section 3.1 to arrive at a preliminary value of six 24-msec time steps for the purposes of these experiments.

For the first run, we tried the simpler and presumably inferior alternative in each of the three dimension of our spectrogram post-processing rule space. Thus the training and testing environments were constructed using linear frequency bands, casewise input scaling, and no shaping function. We ran the batch version of the back-propagation learning algorithm on the training set, gradually increasing the learning parameters from

[3]A network's environment is the collection of input-output patterns that specify the function that the network is supposed to compute.

[4]Strictly speaking, the convergence theorems for 2-layer networks don't apply when a perfect solution doesn't exist (our training set contains conflicting evidence,) but in practice, multiple runs from different starting points always converge to the same solution.

	train		test	
epochs	mean-err	%correct	mean-err	%correct
200	.090	87%	.116	80.3%
400	.066	92%	.115	81.8%
600	.057	94%	.120	81.6%
800	.051	94%	.126	81.6%
1000	.047	95%	.131	81.3%

Figure 2-2:
Performance of a 2-layer network on sub-optimally processed spectrograms containing 6 time steps.

initial values of $\{\varepsilon=.001\ \alpha=.5\}$[5] to a peak of $\{\varepsilon=.005\ \alpha=.95\}$. The weights were sampled after every 200 iterations, and the network's performance was measured on both data sets. Performance was measured in two ways, by computing the mean squared error per case of the output units (relative to their target values)[6], and by counting the number of cases that the network classifies correctly according to the "best guess" rule, which states that the network is voting for the word whose output unit is most active.

The results of this run are shown in figure 2-2. According to both the mean squared error and the best-guess metrics, the network doesn't perform nearly as well on the test set as on the training set. Furthermore, the network's performance on the test set starts to degrade after 400 iterations, while the performance on the training set continues to improve. This is because that the network is straining to learn the training cases in ideosyncratic ways that don't embody general principles of the task.

With this figure of 82% peak generalization in mind, we tried the other spectrogram post-processing possibilities in various combinations, and found that as expected, the alternate method was better in every case. Thus, our final spectrogram format used mel-scaled frequency bands, global (vs. casewise) energy normalization, and input values reshaped with the square function. When a 2-layer network was trained on 6 by 16 spectrograms processed in this manner, it was able to correctly classify 86% of the test cases. The first two processing rules were especially important: a linear frequency scale drops this figure to 83%, as does casewise normalization. Not squaring the values in the spectrograms reduces the performance to 85% generalization.

2.5 Determining the optimal input size

The number of components in a network's input pattern affects the number of weights in the network, which in turn affects the network's information capacity and hence its ability to learn and generalize from a given number of training cases. Using the rule of thumb described in section 3.1, we guessed that the 372 training cases in our possession were only sufficient to train a network with around 500 weights, which

[5] ε is the factor by which the gradient is multiplied, and α is a momentum term.

[6] The target value for an output unit that should be turned off is 0.2, and the target value for an output unit that should be turned on is 0.8. These values eliminate the necessity for large weights to drive the units' activations out to the flat extremes of the sigmoid function.

indicated an optimal input width of less than 8. However, this argument doesn't take into consideration the specific properties of the domain. For example, it is easy to imagine that compressing to a small number of wide time steps could destroy all traces of some important but fleeting articulatory event.

input width	peak generalization
6	71.0%
12	73.5%
24	71.0%
48	70.7%

Figure 2-3:
Generalization performance of a 2-layer network as a function of input width. This is using a strict counting rule that requires the correct ouput unit to be more active than 0.5 and the other three to be less active than 0.5.

To test the validity of our estimate, we generated spectrograms with a range of different time resolutions (6, 12, 24, and 48 time steps), and then used them to train appropriately sized 2-layer networks. Because the bigger networks seemed to have an excessive number of weights, weight decay was used to give them a better chance of staying in the game long enough to exploit the finer detail available to them. For all of the networks, we employed peak parameters of about $\{\varepsilon=.005\ \alpha=.95\ \delta=.001\}$, where δ is the factor by which each weight is decayed after each iteration.

It turned out that the networks were indistinguishable using the best-guess metric, which was somewhat surprising considering the large number of weights contained in some of the networks. The explanation for this performance parity is that 2-layer networks aren't very good at table lookup, since they can only memorize orthogonal patterns. Thus an oversized 2-layer networks doesn't suffer as much in generalization as a oversized multi-layer network would. In crder to get some information as to the relative advantages of the various input sizes, we resorted to a tougher, threshold-based accounting rule that only scores a case as correct when the correct output unit has an activation of more than 0.5 and the other three have activations of less than 0.5. The results of these measurements are summarized in figure 2-3. The spectrograms with 12 time steps (each representing 12 msec) were the winner by a slim margin. Similar experiments with more sophisticated network architectures have confirmed that this input size is in fact the best at providing enough resolution while minimizing the number of connections that have to be trained.

2.6 The best 2-layer network

Figure 2-4 shows four sample spectrograms of each word in the 12 by 16 form that turned out to be the most conducive to good generalization. The corresponding 2-layer network, shown in figure 2-5, has 4 output units which represent the 4 different words. Each of the four output units has 192 connections to the input, so the network has 768 weights that need to be tuned. After undergoing 1000 iterations of the back-propagation learning procedure (consuming about 5 minutes of CPU time on a Convex), the network answered correctly on 93% of the training set and on 86% of the test set. This is

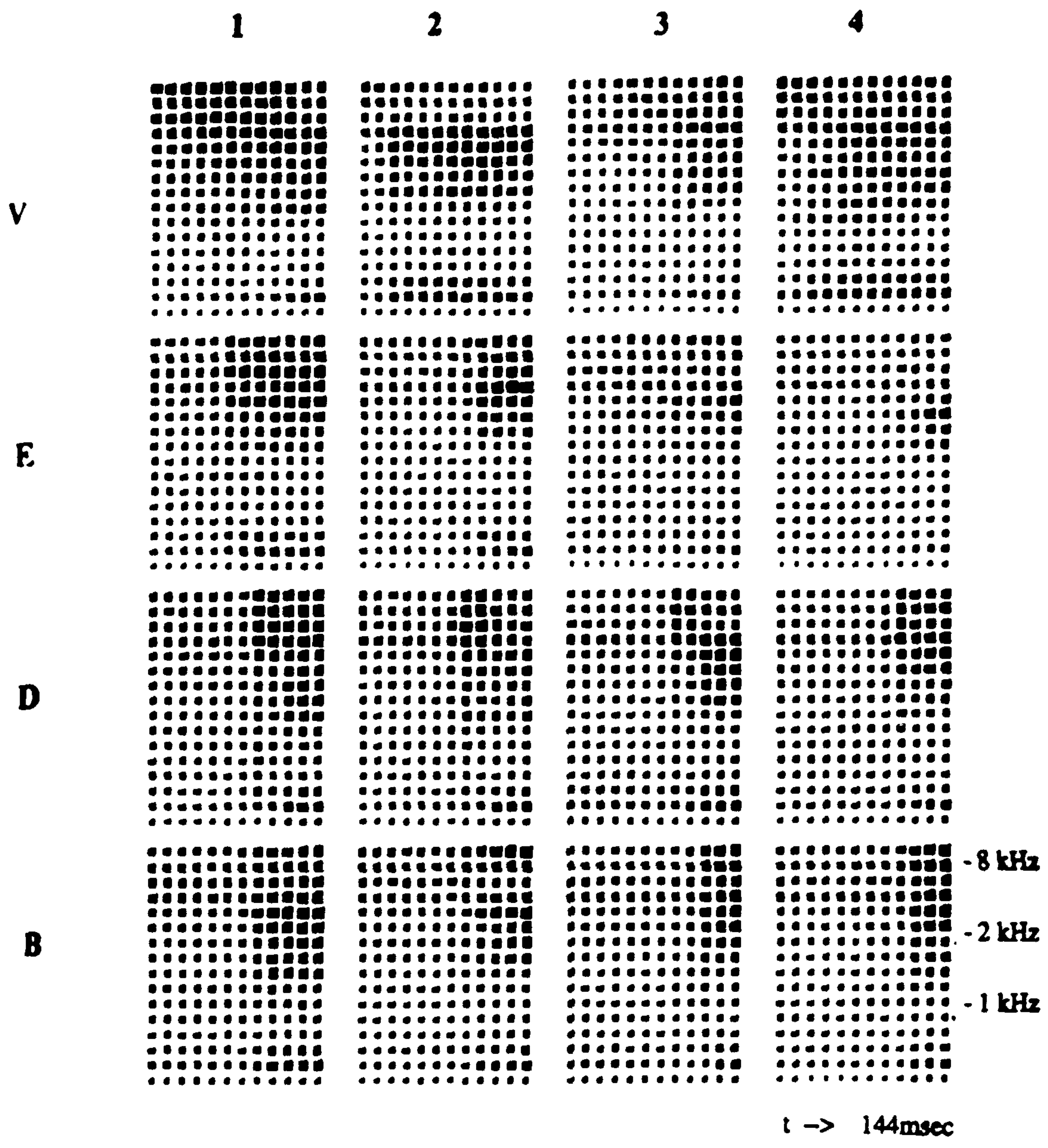


Figure 2-4:
Input spectrograms for 4 samples of each word.

surprisingly close to Peter Brown's best Hidden Markov Model, which scored approximately 89% on the test set.[7]

More interesting than mere performance is the fact that the network has managed to extract sensible looking weight patterns from noisy, hard to read spectrograms. Figure

[7]The 89% performance rating for Brown's model on the BDEV task had to be estimated because his model was actually trained and tested on the full 9 member E-set. We examined the performance statistics of his model for the words B, D, E, and V, and only counted mistakes if the wrong answer was also in the 4 word subtask. Thus a D identified as a T would be counted as correct. We believe that this counting rule offsets the disadvantage of being tuned for a larger task.

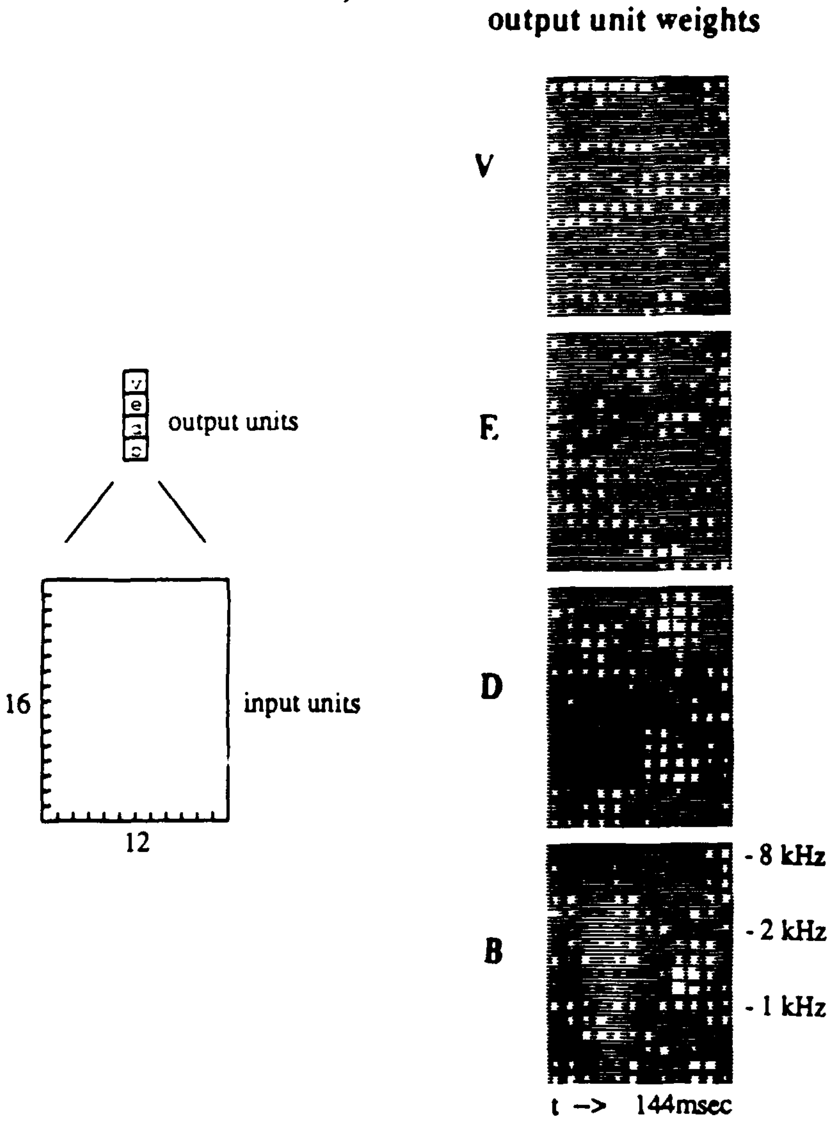


Figure 2-5:
A 2-layer network with 4 output units connected directly to the inputs. The weight patterns on the right contain spectrogram features that the learning procedure has extracted to support each of the 4 answers. White and black blobs represent excitatory and inhibitory weights.

2-5 shows the weights connecting each of the output units to the input spectrogram. The vowel onset can be clearly seen in each pattern near the 9th time slice. When we showed these patterns to Alex Waibel, he pointed out that the D unit is stimulated by a high frequency burst at the vowel onset, while a rising F_2 excites **B** and inhibits **E**. Prevoicing also inhibits **E**, and early frication is evidence for **V** and against **B** and **D**.

There are several reasons to be dissatisfied with this simple network. Its weight patterns require the input to be precisely aligned in time, reducing the network to the status of a template matcher that would be hard to use in the world of continuous speech recognition. Moreover, the network has no hidden units and is therefore of limited power. The networks described in the remainder of this report will correct these deficiencies, and gain additional performance in the process.

3. Hidden Units

3.1 Allocating more training data

Because there were approximately 400 training cases available for this task, each of which requires an output choice that can be specified with 2 bits, a network would need to learn 800 bits of information to perform the task by table lookup. According to CMU folklore, each weight in a network can comfortably store approximately one and a half bits, so a network with more than about 500 weights would have a tendency to memorize the training cases and thus fail to generalize to the test set.

A ceiling of just 500 weights is a serious impediment to the construction of interesting networks, especially since the performance penalty for exceeding the ceiling increases as the sophistication and computational power of a network's architecture grows. The only reliable way to raise this ceiling would be to acquire more training data. We had a total of 767 utterances available, but 396 of them were reserved for testing. As it seemed like overkill to have so many test cases, we transferred most of them to the training set, keeping only 25 randomly selected examples of each word in the test set.

There are three reasons why this redistribution doesn't constitute cheating. Clearly, if this had been a true speaker-independent task, such a move would nullify its status as such. But, the task here was to construct a group model of the 100 speakers based on one utterance per word per speaker, and then to test the model on a second utterance of each word by the same group of speakers. In the modified task, the test set still measures performance on utterances from speakers that it has seen exactly once before. However, the smaller test set doesn't require any knowledge of the speakers that it has seen twice before in the training set. As far as speaker identity is concerned, the task has theoretically been made harder, not easier.

To ensure that the new randomly selected test set was a fair sample of the full set, we performed a quantitative comparison of the relative difficulty of the old and new test sets. Taking the network of figure 2-5, whose weights had been learned on the original, 372 case training set, we measured the error rates on the two test sets. On the full test set the network had a mean squared error of .100 per case, and got 73% of the cases correct using the strict, threshold-based counting rule, and 86% correct using the best-guess counting rule. On just the subset of the test cases, the network had an mean squared error of .116, and only got 68% and 84% of the cases correct according to the two counting rules. Thus, the new test set is slightly more difficult than the old.

The final consideration was one of fairness to the system that we were specifically competing with: the HMM with continuous acoustic parameter modeling that Peter Brown had developed for his thesis. Although our new test set is harder than the old, we wouldn't be able to compare results after increasing the size of our training set if it weren't for the fact that Brown used a similar trick to increase the size of his training set. He employed the same sort of 7/8 data split, but repeated the experiment 8 times with different subsets and then averaged his performance figures to eliminate all artifacts of having a small test set. Rather than go to all of that trouble (burning two months of Convex time in the process), we only used one testing subset, but given the evidence that our subset is a bit harder than necessary, we believe that our results can be reasonably compared to Brown's.

3.2 Adding a hidden layer

Although the 86% generalization achieved by our 2-layer network was respectable, it didn't measure up to the 89% figure established by Peter Brown's best Hidden Markov Model. The problem with a 2-layer network lies in its limited representational power; the simplistic models computable by the network were only able to represent 93% of the training cases in a principled manner that would carry over to the test set.

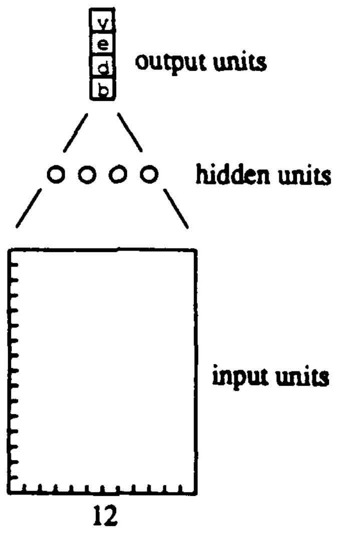


Figure 3-1:

By adding layers to a back-propagation network, one can increase the computational power of the network. Hopefully the expanded family of possible functions will then permit a more natural fit to the training data. To measure the benefits of an additional layer with minimal side-effects, we build and trained a 3-layer network that had a 12 by 16 array of input units, 4 hidden units, and 4 output units.

This network, shown schematically in figure 3-1, is not very different from the 2-layer network discussed in section 2. The two networks have nearly the same number of connections (792 vs. 772), and in their second layers, both networks are forced to represent all of the relevant information using only four activation values. The similarity between the two networks is reflected by their similar learning trajectories, which are summarized in table 3-2. The distinguishing characteristics of the 3-layer version are a longer training time, the ability to learn more of the training cases, and slightly better test scores according to the mean squared error and threshold-based metrics, which both have more of an analog character than the best-guess metric. The additional layer helps the 3-layer network squeeze its outputs closer to their target values of 0.2 and 0.8 on each case, regardless of whether the rank order of the various answers is correct.

The second network that we tried had a better chance for improved performance. This network contained twice as many (8) hidden units, which doubled both the network's information capacity and the bandwidth of its hidden layer. The learning trajectory of this network is shown in the left half of table 3-6. The network immediately consumed the training set, mastering 99% of the cases in just 2400 epochs. At that point, the network was able to correctly classify 89% of the test cases. An examination of the

2 - layer no hidden units					3 - layer four hidden units				
	train	test				train	test		
hits	bges	squer	thres	bges	hits	bges	squer	thres	bges
200	90	.120	38	20	2000	97	.113	33	18
400	53	.117	31	16	4000	49	.110	25	16
800	45	.120	30	14	6000	16	.128	29	14

Figure 3-2:
A comparison of the learning trajectories of a 2-layer network and a 3-layer network with 4 hidden units. This table shows the error rates of the networks on the training and testing sets at selected times during the training process. Three error metrics were used: the mean squared error per case, and the number of erroneously classified utterances (out of 667 or 100 cases for the training and testing sets respectively) according to the threshold-based and best-guess accounting rules. The threshold-based rule requires an output unit to have an activation above 0.5 if and only if it represents the correct word. The best-guess rule considers the network to be correct when the output unit representing the correct answer is more active than the others.

network's weights, which are pictured in figure 3-3, shows that the network used four of its hidden units as templates for the four words (much like those developed by the 2-layer network). The remaining hidden units represented the disjunctions {**BD**},{**BE**}, and {**EV**} and an alternate form of **D**.

3.3 Receptive fields

The 3-layer networks described in the previous section are of the unsophisticated "bag-of-hidden-units" variety. Every hidden unit is connected to all of the input units and to all of the output units. The weight patterns of figure 3-3 show that each hidden unit tends to form an overall spectrogram template for one or more of the words.

According to the standard intuitive explanation of the behavior of multi-layer feed-forward networks, hidden units are supposed to extract meaningful features from the input patterns that are presented to a network. These features are then passed on as evidence for the output units to consider as they decide on the network's answer. The intuitive notion of a spectrogram feature generally involves a localized sub-pattern in the spectrogram. One can force a network to develop localized feature detectors by restricting its connectivity, giving each hidden unit a receptive field that only covers a small subset of the input.

A network with small receptive fields has a couple of significant advantages over fully connected networks. Because the total number of its weights is a small multiple of the number of its hidden units, a receptive field network enjoys a large ratio between the information bandwidth of the hidden layer and the total information capacity of the network. Thus the network can possess a rich inventory of hidden layer codes to represent subtleties of the input without being burdened with an excessive number of free parameters that would allow the network to learn its training set by rote. The second

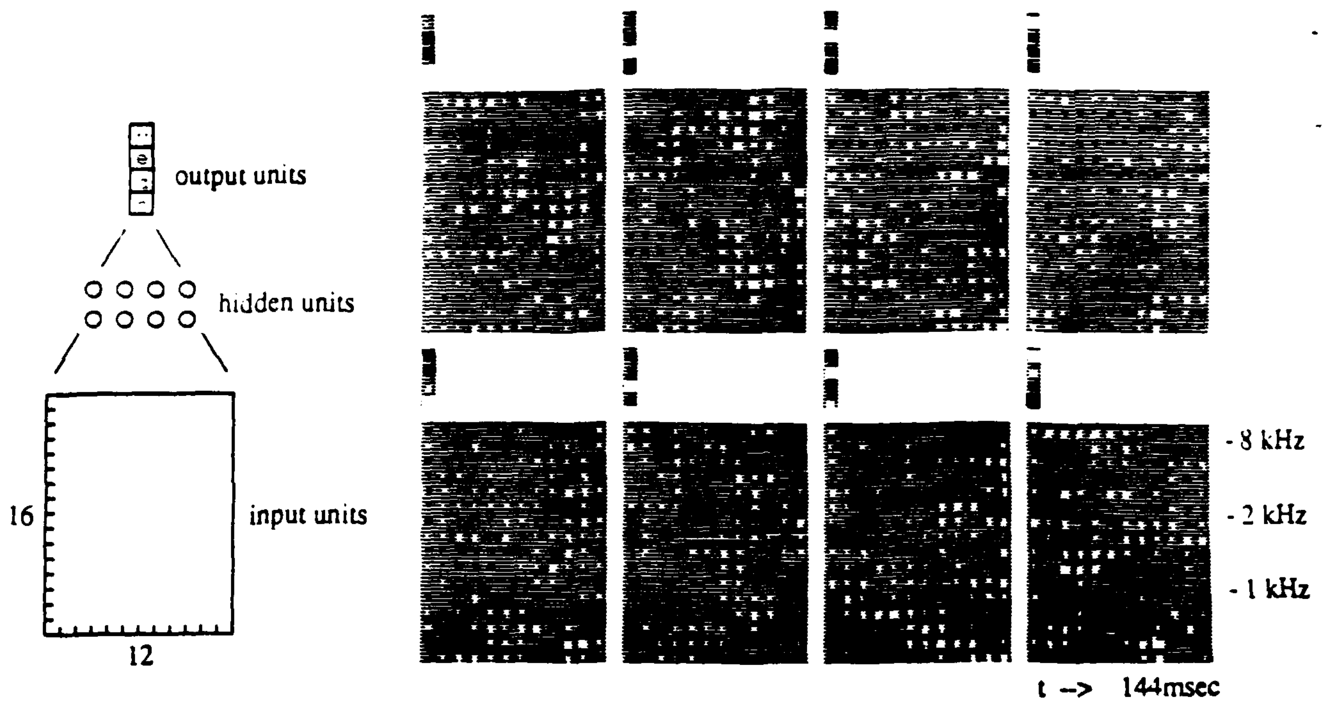


Figure 3-3:
A 3-layer network with 8 hidden units. The 8 patterns on the right show the weights learned by each of the 8 hidden units. Each pattern includes connections to the 12 by 16 input array and to the 4 output units.

advantage of this sort of network is that the hidden units are automatically assigned unique roles in solving the task. Because the various hidden units are looking at different portions of each input pattern, there is no chance that the back-propagation training procedure will give them duplicate weight patterns.

To demonstrate the benefits of this network architecture, we constructed a receptive field network with approximately the same number of weights as the 8 hidden unit network discussed in the last section. As shown in figure 3-4(a), each hidden unit is connected to a slice of the input spectrogram that contains only 3 time steps (but all 16 frequency bands). Since there are 10 ways to position a 3-step window on a 12-step input, the input is covered by 10 different time slices. To permit the detection of multiple features in each slice of the input, the network has 3 separate hidden units connected to each of the 10 receptive fields, for a total of 30 hidden units. The 4 output units are connected to all 30 hidden units, so the network contains $30 \times 3 \times 16 + 4 \times 30 = 1560$ weights. After training, these weights assumed the values shown in figure 3-5.

The learning trajectories contained in table 3-6 show that this network performed slightly better than the fully connected network with 8 hidden units according to the mean squared error and threshold-based metrics. Although the receptive field architecture didn't provide a big improvement over the fully connected architecture on the BDEV task, it has proven to be clearly superior on tasks that require to network to discriminate between consonants in a variety of vowel contexts, in which case it is useful for the network to be able to represent information about different parts of the spectrogram using separate hidden units.

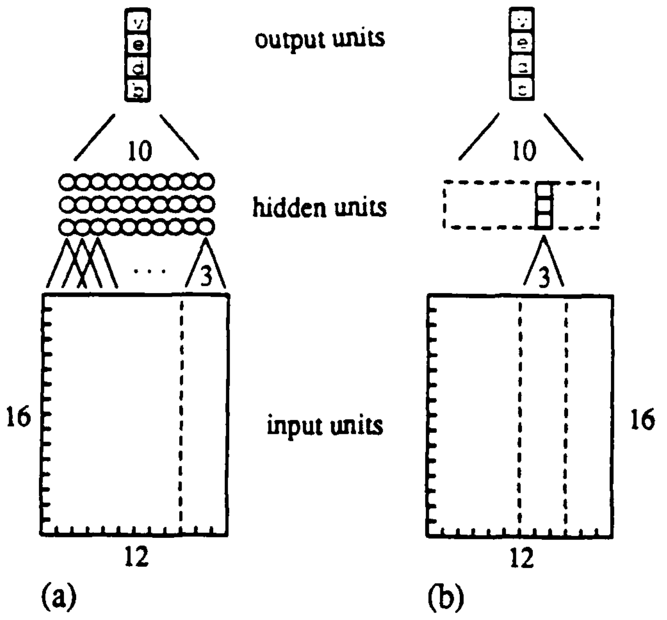


Figure 3-4:
Two views of a 3-layer network with 30 hidden units that is each connected to a window of 3 time steps.

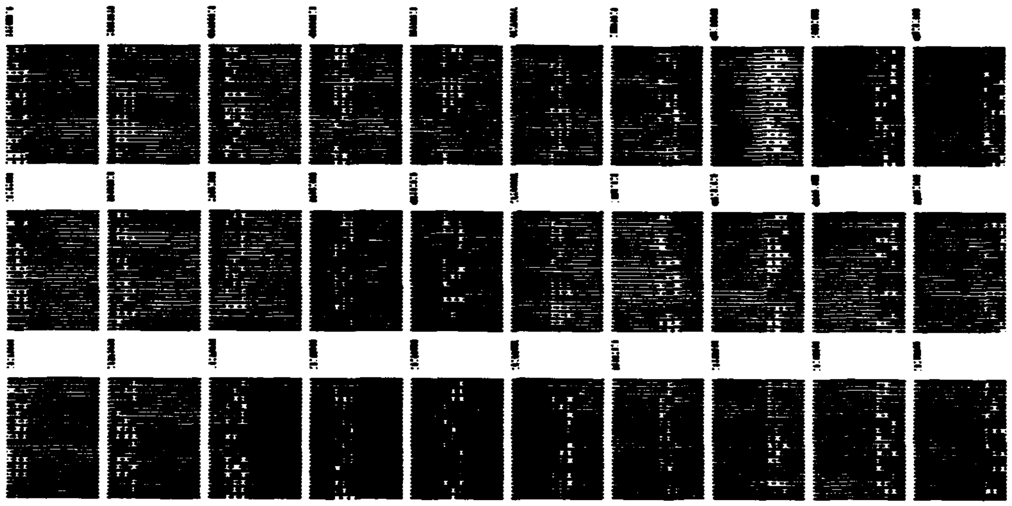


Figure 3-5:
The weights of the 30 hidden units from the 3-layer receptive field network diagrammed in figure 3-4(a).

3.4 Position independent feature detectors

In the receptive field architecture described in the last section, the hidden units are all free to develop weight patterns for detecting the features that are most relevant to the particular portions of the words that lie in the units' receptive fields. Figure 3-5 shows that the feature detectors developed at each time slice are in fact different from the

8 fully connected hidden units					30 narrow receptive field hidden units				
	train	test				train	test		
hits	bges	squer	thres	bges	hits	bges	squer	thres	bges
800	65	.110	32	18	1000	88	.120	38	21
1200	43	.115	25	14	2000	43	.108	24	16
1600	23	.118	21	13	3000	27	.110	24	13
2000	16	.116	21	14	4000	17	.110	21	13
2400	9	.112	21	11	5000	9	.109	20	12
2800	6	.115	21	14	6000	6	.110	21	11

Figure 3-6:
A comparison of the learning trajectories of two 3-layer networks that both contain about 1500 weights. In the first network, all of the hidden units are connected to the entire input. In the second, each hidden unit is connected to a window of 3 time steps out of 12. See figure 3-2 for an explanation of the contents of this table.

feature detectors at every other time slice. This freedom to develop specialized hidden units for analyzing the various parts of the words would be desirable if all of the exemplars of the words had exactly the same alignment relative to the 12-step input array. However, when a misaligned word is presented as input, the fact that these specialized detectors are hard-wired to the input array means that the wrong detectors will be applied to the wrong parts of the word. One way to eliminate this problem is to force the network to apply the same set of feature detectors to every slice of the input.

A small modification to the back-propagation learning procedure is required to make a receptive field network act in the desired manner. Consider the network of figure 3-4(a), which contains 3 rows of 10 hidden units connected to 10 successive 3×16 windows into the input. The 10 weights connecting the hidden units of a given row to the 10 successive input units representing a given receptive field component[8] are thrown into an equivalence class. After the weights are updated at the end of each iteration of the learning procedure, every weight in an equivalence class is set to the average of the weights in that class. When the network is trained using this rule, all of the hidden units in a given row will have learned the same weight pattern, so the row can be thought of as a single hidden unit replicated 10 times to examine 10 successive input slices for the presence of one feature.

This new interpretation of the receptive field network is shown schematically in figure 3-4(b). The network effectively contains only 3 different hidden units. Because each hidden unit is connected to the input units via a 3 by 16 weight pattern, there are $3 \times 3 \times 16 = 144$ weights between the first and second layers. Although the 10 copies of a given hidden unit possess identical weights, they assume 10 different activation levels that represent the presence or absence of the unit's feature in the 10 slices of the input. Since the activation level of each copy of a hidden unit conveys information about the nature of the input, every copy gets its own connection to the output layer. Thus there are

[8]*e.g.* the upper left-hand corner of the window

$4 \times 10 \times 3 = 120$ weights between the second and third layers.

Now, because 3 hidden units aren't enough to reasonably hope for good performance on this task, we didn't actually use the network of figure 3-4(b). Experiments with networks containing 4, 6, and 8 replicated hidden units showed that a network with 8 hidden units works the best. Figure 3-7 is a picture of the activation levels of the $8 \times 10 = 80$ hidden unit copies of a replicated network on the same 16 utterances whose input spectrograms were exhibited back in figure 2-4. These hidden unit activation patterns can be thought of as pseudo-spectrograms that have arbitrary features rather than frequency band energies displayed on the vertical axis. One can see that each of the four words has a characteristic feature pattern, and that for a given utterance, the time alignment of the word in the original spectrogram is preserved in the corresponding pseudo-spectrogram (compare the first and fourth **B** and **D**.) This means that time alignment errors are passed right on through the hidden units to the fixed-position output units, which are still unable to classify misaligned utterances. A solution to this problem, namely the replication of output units, is presented in the next section.

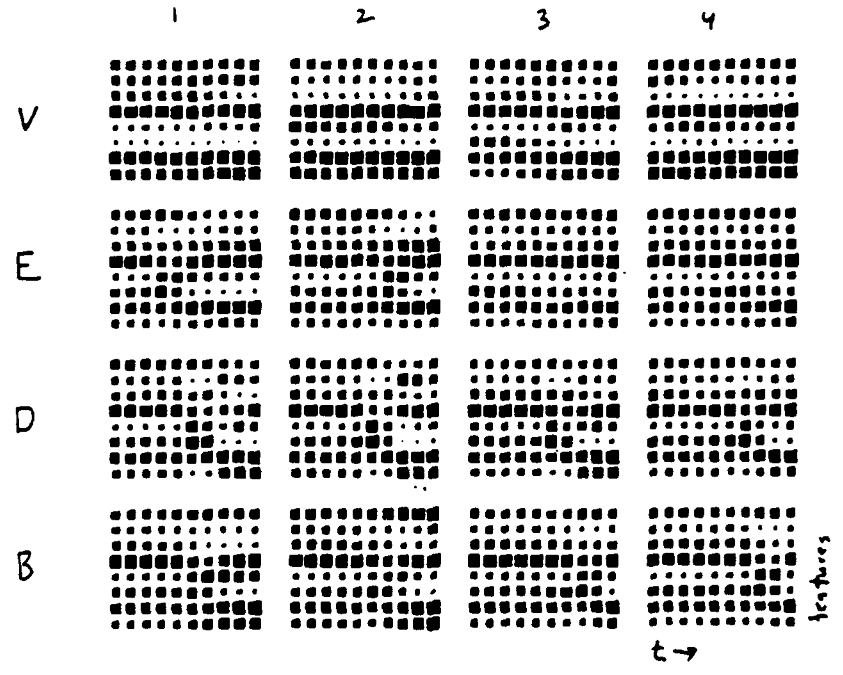


Figure 3-7:
Hidden unit activation patterns from a 3-layer network with replicated hidden units on 4 samples of each word. The 10 copies of 8 hidden units show the presence or absence of 8 features for 10 successive time positions.

4. Networks with Built-in Time Symmetry

4.1 A 2-layer network with replicated output units

An analysis of the errors made by the 3-layer networks of section 3 showed that the most common source of error was incorrect alignment of the word on the input frame. Because the position of the vowel onset in each utterance was chosen by a mostly successful Viterbi alignment, there aren't enough different starting points in the training data to allow a network to learn to generalize across time.

To solve this problem, we devised a network architecture that explicitly captures the concept of time symmetry. Networks of this sort contain several copies of each output unit. The various replicas of a unit apply the same weight pattern to each of several narrow slices of a spectrogram, searching for an activation pattern characteristic of the word denoted by the unit. During learning, the equivalence class rule described in section 3.4 constrains the weight patterns of all of the copies of each output unit to be the same.

As with all of the networks described in this report, the 2-layer network shown in figure 4-1 has 4 output units. Whereas the output units of the simple 2-layer network of figure 2-5 were connected to the entire input, the output units in this network are connected to narrow receptive fields that only cover 5 time steps. Since there are 8 ways to position a 5 step window on a 12 step input, the network contains 8 copies of each output unit. When an input is presented to the network, each of the $4 \times 8 = 32$ output unit copies is activated by an amount that indicates the copy's confidence that its word is present, based on the evidence that is visible in its receptive field. The value of an output unit is defined to be the sum of the squares[9] of the activations of all of the copies of that unit, so the overall computation performed by the network is a mapping from spectrograms to real valued 4-tuples, as always.

The 4 output unit weight patterns shown in figure 4-1 were those arrived at after 11,000 iterations of the learning procedure. With these weights, the network correctly classified 94% of the training cases and 91% of the test cases. This is a significant improvement over the 86% generalization achieved by the fixed position 2-layer network of figure 2-5. A comparison of the weight patterns learned by the two networks is illuminating. In the new network, the rising F_2 of the **B** pattern and the high-frequency burst of the **D** pattern are cleanly localized in time, while in the old network, these events were smeared over two or three time steps. Because the replicated network has time symmetry built into its architecture, it no longer has to compensate for variable word alignment in its weight patterns, thus allowing the network to analyze the critical portions of the spectrograms in more detail.

[9]The motivation for squaring the activations was to allow the activation of the output unit replica that found the best match to predominate in the overall answer. To find out whether this effect was really beneficial, we trained a toy network consisting of 5 input units and 2 replicated output units to distinguish between the patterns 101 and 110 regardless of the patterns' alignment on the 5 input units. Using the squared activation rule, the network took 311 iterations to learn the task, employing weights whose average size was 0.99. When we tried the same task using the sum of the output units' unsquared activations as the network's outputs, the network needed 558 iterations and weights of average size 1.23 to solve the problem.

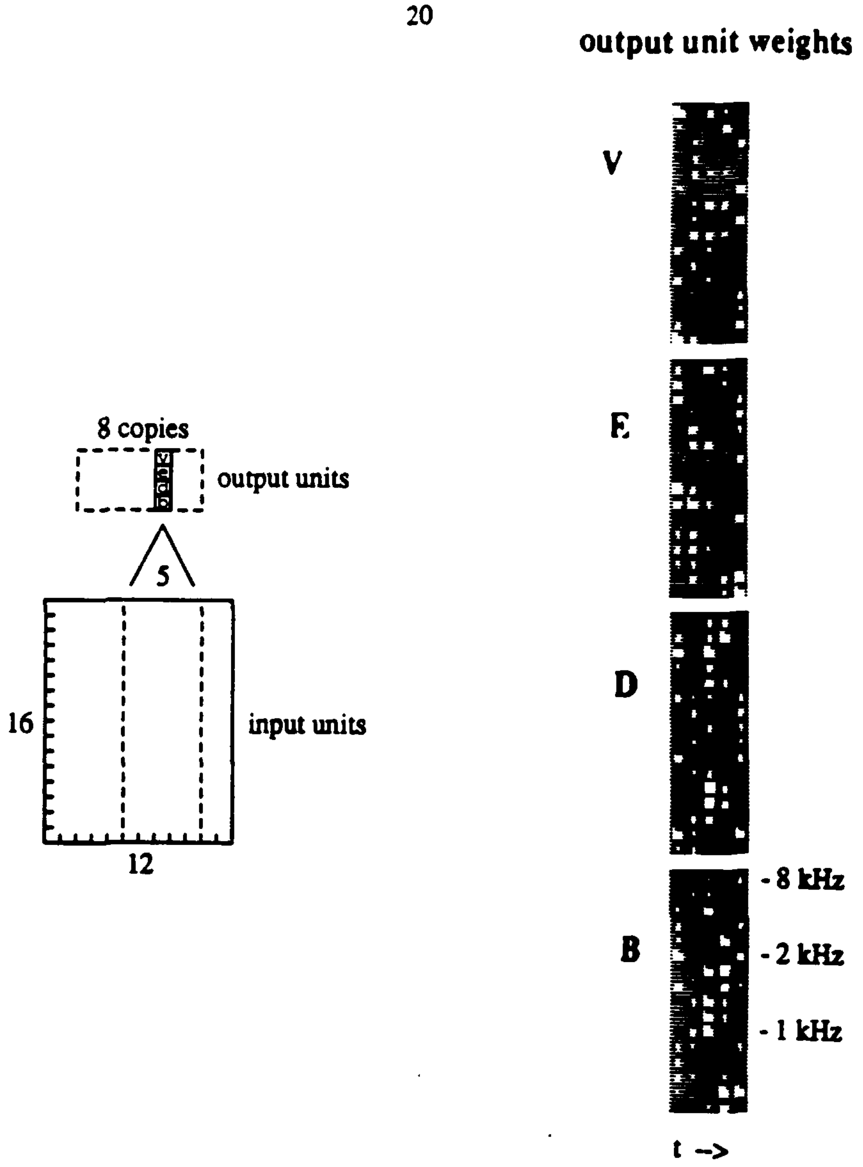


Figure 4-1:
A 2-layer network whose output units are replicated across time. The weight patterns shown on the right are applied by 8 copies of the 4 output units to successive 5-step windows into the input. White and black blobs represent excitatory and inhibitory weights.

4.2 A 3-layer network with replicated output units

The experiment described in section 4.1 demonstrated that even without hidden units, a network using our technique for factoring out the effects of time alignment was able to perform better than the best hidden markov model developed for this task. Hopefully, the addition of some hidden units would result in a network with even better performance. As before, the first layer of the 3-layer replicated network (see figure 4-2) consisted of 192 input units encoding a spectrogram. The hidden layer contained 10 copies of 8 hidden units that were each connected to a 3-step slice of the input. The third layer had 6 copies of the 4 output units, each looking at a 5-step slice of the pseudo-spectrogram generated by the hidden layer.

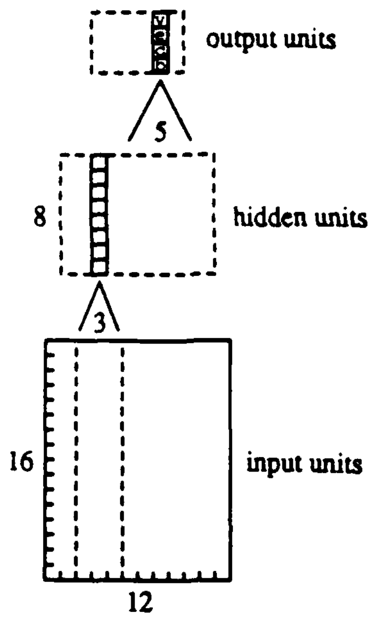


Figure 4-2:
A 3-layer iterative network whose hidden units and output units are replicated across time.

The weight space of a highly constrained multilayer network is more difficult to explore than that of a simpler network, requiring smaller and more carefully chosen learning parameters. More than 20,000 iterations with peak parameters of {ε=.001 α=.95} were needed to tune the network into a model that accounted for 93% of the training cases, and 93% of the test cases. The activation patterns of this network's output units on 16 sample utterances are pictured in figure 4-3. The detectors for **E** and **V** show little time locality, utilizing features that are globally present in the words. However, the network recognizes the stops **B** and **D** by detecting the vowel onset, so the activation traces for them clearly show the alignment of the utterance. Notice that the **B** detector fires slightly later on examples three and four, as one would expect from examining the input spectrograms (see figure 2-4.)

It is significant that the network learned to locate and analyze the consonant-vowel transition point, despite the fact that the training environment didn't include any explicit information about the usefulness of this region of the word, much less any information about where the vowel onset could be found in a given utterance. The network's success at learning to find and exploit the most informative region of each spectrogram suggests that the Viterbi alignment initially used to clip 144 msecs from each utterance was unnecessary; the network might have done just as well if it had been shown the entire words with completely unknown alignment.

The fact that the network manages to get the same performance (93%) on the training and test sets suggests that the underlying structure of the training corpus really is being modeled here, rather than peculiarities of specific cases. This may expain why traditional connectionist tricks for improving generalization, such as weight decay and adding noise

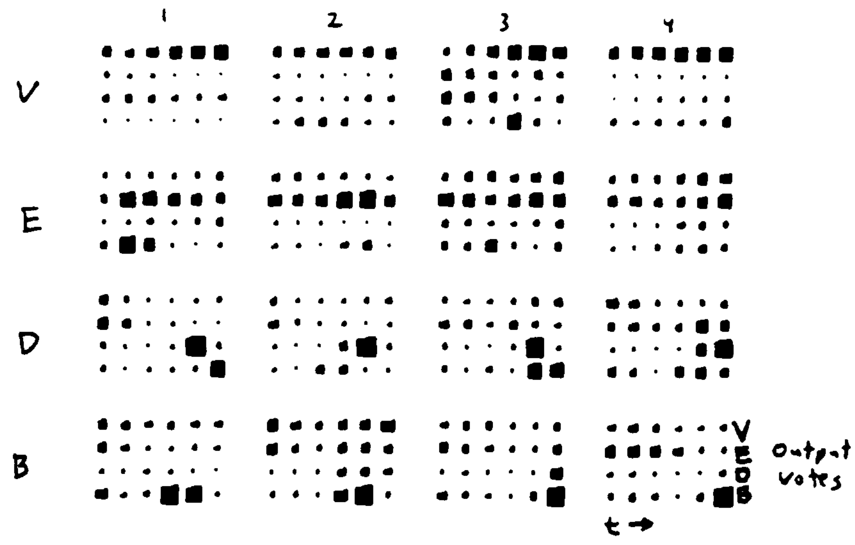


Figure 4-3:
Output unit activation patterns from a 3-layer replicated network on 4 samples of each word. These show the confidence of the output unit copies corresponding to successive time positions. Observe how the **D** and **B** detection events are localized in time.

to the training cases, haven't improved the network's performance. Moreover, it suggests that this group model of 100 speakers in 544 weights might attain similar performance as a true speaker independent **BDEV** recognizer.

From a connectionist point of view, one important lesson of this work is its demonstration of the value of building knowledge into the structure of a network before training it. Although back-propagation could have taught the network about time symmetry if there had been enough examples of words in different positions, the training data was deficient in this respect. In the real world, there is usually a scarcity of training data, so it is important to use techniques that allow knowledge to be built in as well as learned.

4.3 An important derivative

So far, we have glossed over the question of how one would go about training a network with replicated output units. The error of a back propagation network on a given case is a function of the differences between the network's actual output values o_j and the corresponding target values d_j.

$$E = \frac{1}{2}\sum_j (o_j - d_j)^2$$

In ordinary back-propagation, the output values o_j of a network are just the activation levels of its output units y_j. Plugging this fact into the definition of E and then differentiating by y_j gives us the partial derivative of the error with respect to the activations of the output units. These values provide the starting point for the backward pass of the learning algorithm.

$$\frac{\partial E}{\partial y_j} = y_j - d_j$$

In our replicated network architecture, each output value of the network is the sum of the squares of the activations of several temporal replicas of an output unit.

$$o_j = \sum_t y_{jt}^2$$

Plugging this into the definition of E and then differentiating yields the partial derivative of the error with respect to activation of the replica of unit j at time τ.

$$\frac{\partial E}{\partial y_{j\tau}} = 2y_{j\tau}((\sum_t y_{jt}^2) - d_j)$$

4.4 Time-delay neural networks

Our architecture was originally conceived in terms of replicated networks trained under constraints which ensured that the multiple copies of each unit applied the same weight pattern to each part of the spectrogram [Lang 87]. Since the constrained training regime for a replicated network is similar to the standard technique for simulating iterative back-propagation networks [Rumelhart 86], it is tempting to re-interpret the replicated network in iterative terms [Hinton 88]. According to this viewpoint, the network has 16 input units, 8 hidden units, and 4 output units (see figure 4-4). Each input unit is connected to each hidden unit by 3 different links having time delays of 1, 2, and 3. Each hidden unit is connected to each output unit by 5 different links having time delays of 1, 2, 3, 4, and 4. The time delay nomenclature associated with this iterative viewpoint was adopted by the researchers at ATR who have been experimenting with our network architecture [Waibel 87].

4.5 Related work

The idea of replicating network hardware to achieve position independence is an old one [Fukushima 80], and a training procedure for networks with temporally replicated units was presented in the original back-propagation paper [Rumelhart 86]. In our network architecture, the replicated output units occupy a role that lies somewhere between those of traditional hidden units and traditional output units. Particular target values for the individual output unit copies are not specified externally; the only thing

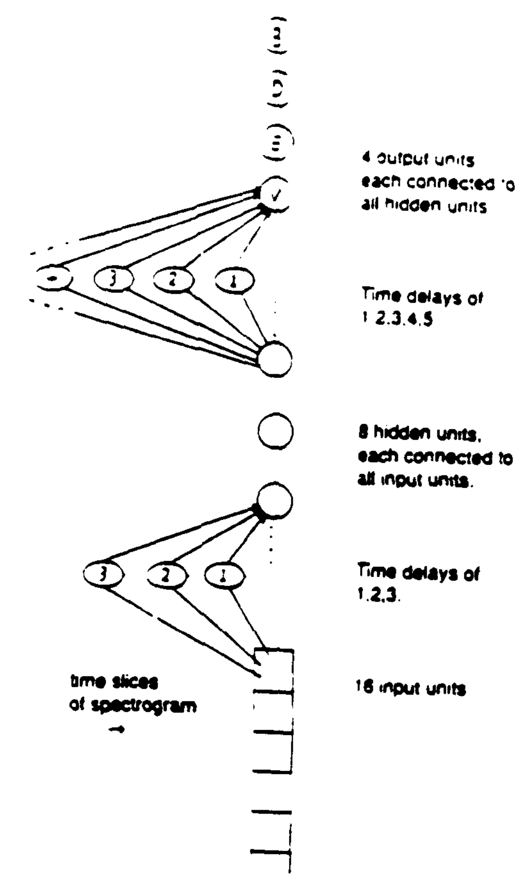


Figure 4-4:
The 3-layer replicated network of figure 4-2
re-interpreted in terms of time delays.

that is specified is the value that a function of their combined activations should assume. Thus our architecture is an instance of the family of back-propagation networks with post-processing functions that David Rumelhart has been investigating.

When viewed as an iterative network with time delays, our architecture has some similarities to the one proposed in [Tank 87]. The distinguishing characteristics of our work are the use of the back-propagation learning rule and the use of a post-processing function for combining evidence from the various time steps.

5. A Multi-Resolution Training Technique

5.1 The idea

In an effort to improve the 93% performance achieved on the BDEV task by the time-delay neural network described in the last section, we retrained the network using a couple of traditional connectionist techniques for enhancing generalization performance, namely decaying the weights and adding random noise to the inputs during training. Unfortunately, both of the techniques hurt rather than helped the network's performance. After this setback we decided to try a different approach that might be more suited to the nature of the domain.

As you may recall from section 2, the major problem with our original input data was that the spectrograms contained too much detail. A network capable of handling these spectrograms would contain a large number of weights that would take an extremely long time to train. Furthermore, overly detailed spectrograms permit a network to notice and remember idiosyncrasies of individual training cases. Our original solution for this problem was to compress much of the detail out of the inputs, reducing a 48×128 raw spectrogram to a 12×16 input pattern.

A different solution came to mind when we viewed the task as a vision problem; after all, humans read spectrograms using the pattern recognition powers inherent in their visual systems. We hoped that a new training technique inspired by the multigrid relaxation algorithm of [Terzopoulos 86] would force a network to learn the gross structure of the spectrograms before focusing on smaller details, thereby speeding up learning and improving generalization. The proposed technique consisted of training a sequence of increasingly large networks to classify increasingly detailed spectrograms. Once a network had been trained, each of its weights would be used to initialize a group of finer-grained weights in the next network. During the learning process, neighboring weights would be constrained to be similar by a cost function that would allow the network to focus on fine details when they were important, but encourage the network to ignore them when they were not.

5.2 Summary of results

We evaluated the merits of the proposed multi-resolution scheme by training variously sized 2-layer networks and 3-layer TDNN's both with and without the weight transfer technique and the various versions of the cost function. Since most of the experiments had negative results and are therefore somewhat uninteresting, only the overall conclusions will be presented here.

The smoother, less detailed weight patterns learned under the influence of a cost function turned out to be of benefit only to oversized networks (containing more than 12 time steps) whose performance was poor to begin with; the performance of 12-step networks declined when they were forced to use smoother weight patterns.

The second component of the multi-resolution training regime had more utility. This was the technique of starting network off with initial weights derived from a lower resolution starter network. For 2-layer networks, the weight transfer technique still only improved the performance of oversized networks, but 3-layer networks benefited even when they were of optimal size. We noticed that 3-layer multi-resolution networks used their hidden units more efficiently than the same networks trained from random initial weights. This unanticipated side effect of the weight transfer technique yielded a 12-step time delay network that was able to correctly classify 94% of the test cases. By comparison, the best hidden Markov model in Peter Brown's thesis achieved approximately 89% generalization, and human listening performance on this data set is 95%.

5.3 The best network for the BDEV task

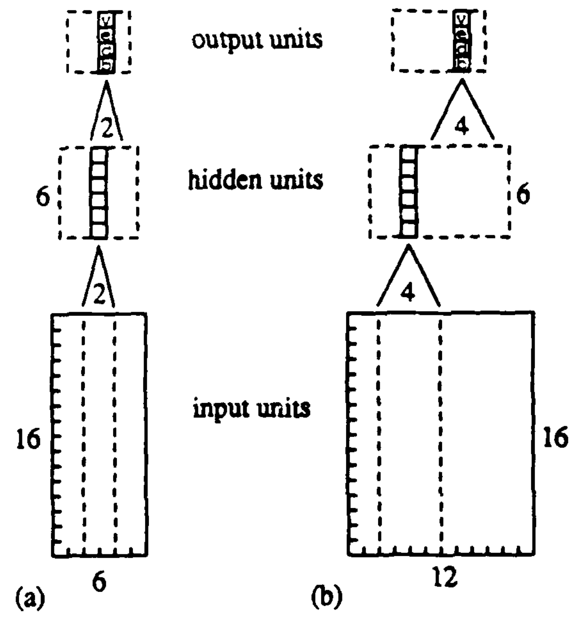


Figure 5-1:
(a) A 6-step starter TDNN.
(b) A 12-step TDNN initialized by network (a).

We trained the 6-step TDNN shown in figure 5-1(a) until it had learned to correctly classify 85% of the training cases. This required 3000 epochs with parameters that were gradually increased to peak values of $\{\varepsilon=.001\ \alpha=.95\}$. At that point, the network's 240 weights, which are shown in figure 5-2(a), had an average magnitude of 0.71.

These weights were then converted into initial weights for the 12-step TDNN shown in figure 5-1(b). This network has the same number of hidden units and output units as the 6-step network, but its input spectrograms and receptive fields are twice as wide. During a weight transfer, each weight in a given unit's receptive field is split into two identical halves.[10] Thus the average magnitude of the initial weights for the 12-step network was about 0.35.

Starting with the half-resolution weights shown in figure 5-2(b), the 12-step network

[10] *i.e.* each weight in a receptive field is duplicated and divided by 2. Bias weights are copied over unchanged.

was trained until its generalization peaked at 13,000 epochs, at which point the network was using the weights shown in figure 5-2(c). Compared to the 20,000 epochs required to train the TDNN of section 4, it is clear that the weight transfer technique did in fact reduce the considerable computational requirements of training these networks. Furthermore, the resulting generalization behavior of the network was better according to all three of our performance metrics than the same network trained without it. The biggest improvement came in the mean squared error, which is an analog measure of the goodness of the fit of the model to the data. According to the best-guess metric, the multi-resolution time-delay network answered correctly on 96% of the training cases and 94% of the testing cases, while the same 12-step network trained from random initial weights peaked at 91% generalization.

After comparing the weight patterns of figure 5-2(c) with similar ones from the network of section 4, we believe that the improved performance of the multi-resolution network is mostly due to its more effective utilization of hidden units, thanks to the valiant efforts of the 6-step starter net, which only had 240 weights available to account for 667 utterances by 100 speakers. The TDNN of section 4 wasted half of its 8 hidden units, while the new multi-resolution TDNN put all 6 of its hidden units to good use. Repeated training runs with these networks have yielded similar results.

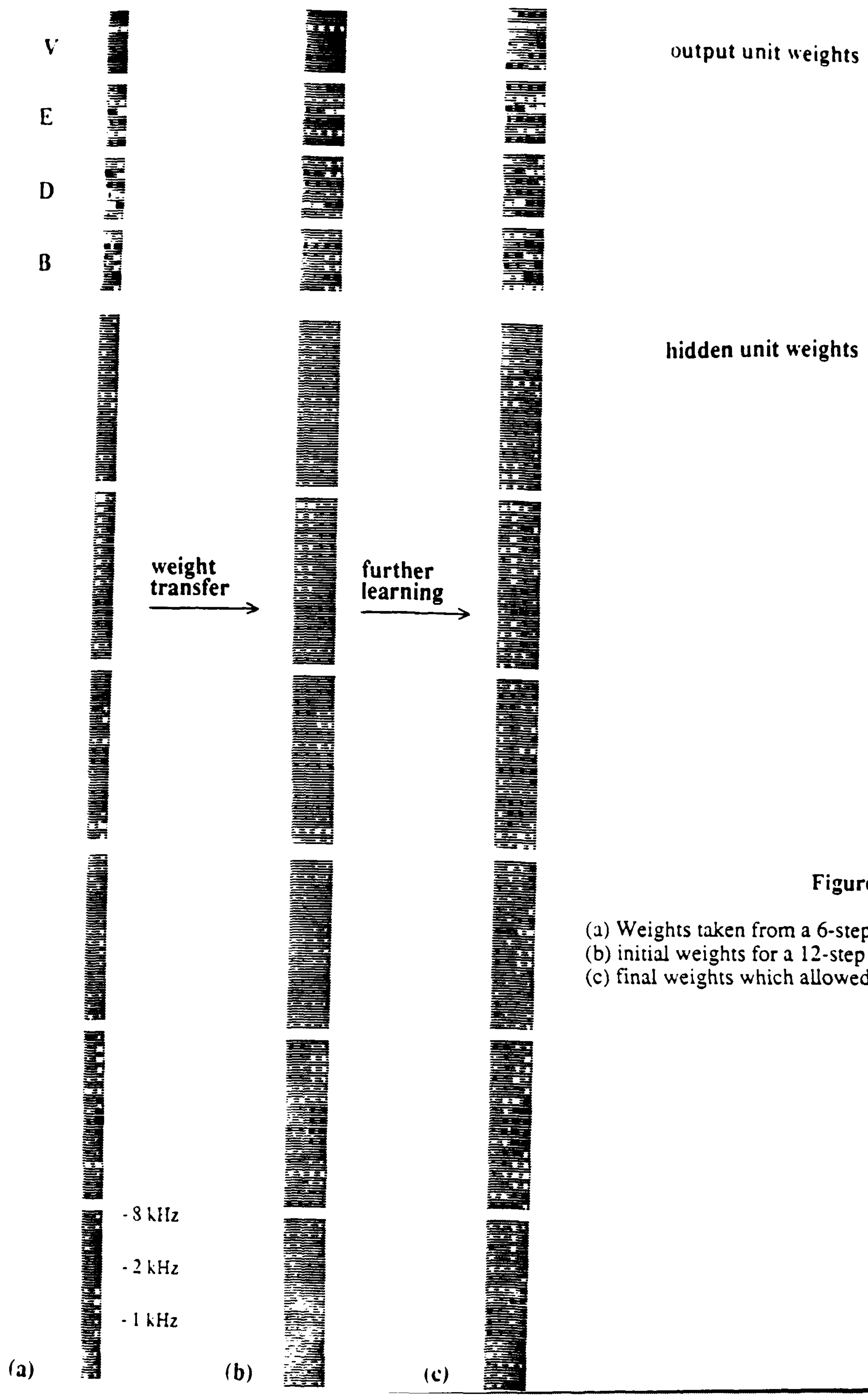


Figure 5-2:

(a) Weights taken from a 6-step TDNN to generate
(b) initial weights for a 12-step TDNN that learned
(c) final weights which allowed 94% generalization

References

[Brown 87]	Brown, P. F. *The Acoustic-Modeling Problem in Automatic Speech Recognition.* PhD thesis, Carnegie-Mellon University, 1987.
[Fukushima 80]	Fukushima, K. Neocognitron: a Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*36, 1980.
[Hinton 88]	Hinton, G. E. Connectionist Learning Procedures. *Artificial Intelligence*, 1988.
[Lang 87]	Lang, K.J. *Connectionist Speech Recognition..* PhD thesis proposal, Carnegie-Mellon University, June, 1987.
[Plaut 86]	Plaut, D. C., Nowlan, S. J., & Hinton, G. E. *Experiments on learning by back-propagation.* Technical Report CMU-CS-86-126, Carnegie-Mellon University, Pittsburgh PA 15213, June, 1986.
[Rumelhart 86]	Rumelhart, D. E., Hinton, G. E. & Williams R. J. Learning representations by back-propagating errors. *Nature*323:533-536, 1986.
[Tank 87]	Tank, D.W. and Hopfield, J.J. Neural computation by concentrating information in time. In *Proceedings of the National Academy of Sciences*, 1987.
[Terzopoulos 86]	Terzopoulos, D. Image Analysis Using Multigrid Relaxation Methods. *IEEE Transactions of Pattern Analysis and Machine Intelligence*PAMI-8(2):129-139, March, 1986.
[Waibel 87]	Waibel, Hanazawa, Hinton, Shikano, and Lang. *Phoneme Recognition Using Time-Delay Neural Networks.* Technical Report TR-I-0006, Advanced Telecommunications Research Institute, Japan, October, 1987.

Table of Contents

LEARNING REPRESENTATIONS BY RECIRCULATION

Geoffrey E. Hinton
Computer Science and Psychology Departments, University of Toronto.
Toronto M5S 1A4, Canada

James L. McClelland
Psychology and Computer Science Departments, Carnegie-Mellon University.
Pittsburgh, PA 15213

ABSTRACT

We describe a new learning procedure for networks that contain groups of non-linear units arranged in a closed loop. The aim of the learning is to discover codes that allow the activity vectors in a "visible" group to be represented by activity vectors in a "hidden" group. One way to test whether a code is an accurate representation is to try to reconstruct the visible vector from the hidden vector. The difference between the original and the reconstructed visible vectors is called the reconstruction error, and the learning procedure aims to minimize this error. The learning procedure has two passes. On the first pass, the original visible vector is passed around the loop, and on the second pass an average of the original vector and the reconstructed vector is passed around the loop. The learning procedure changes each weight by an amount proportional to the product of the "presynaptic" activity and the *difference* in the post-synaptic activity on the two passes. This procedure is much simpler to implement than methods like back-propagation. Simulations in simple networks show that it usually converges rapidly on a good set of codes, and analysis shows that in certain restricted cases it performs gradient descent in the squared reconstruction error.

INTRODUCTION

Supervised gradient-descent learning procedures such as back-propagation[1] have been shown to construct interesting internal representations in "hidden" units that are not part of the input or output of a connectionist network. One criticism of back-propagation is that it requires a teacher to specify the desired output vectors. It is possible to dispense with the teacher in the case of "encoder" networks[2] in which the desired output vector is identical with the input vector (see Fig. 1). The purpose of an encoder network is to learn good "codes" in the intermediate, hidden units. If for, example, there are less hidden units than input units, an encoder network will perform data-compression[3]. It is also possible to introduce other kinds of constraints on the hidden units, so we can view an encoder network as a way of ensuring that the input can be reconstructed from the activity in the hidden units whilst also making

This research was supported by contract N00014-86-K-00167 from the Office of Naval Research and a grant from the Canadian National Science and Engineering Research Council. Geoffrey Hinton is a fellow of the Canadian Institute for Advanced Research. We thank Mike Franzini, Conrad Galland and Geoffrey Goodhill for helpful discussions and help with the simulations.

the hidden units satisfy some other constraint.

A second criticism of back-propagation is that it is neurally implausible (and hard to implement in hardware) because it requires all the connections to be used backwards and it requires the units to use different input-output functions for the forward and backward passes. Recirculation is designed to overcome this second criticism in the special case of encoder networks.

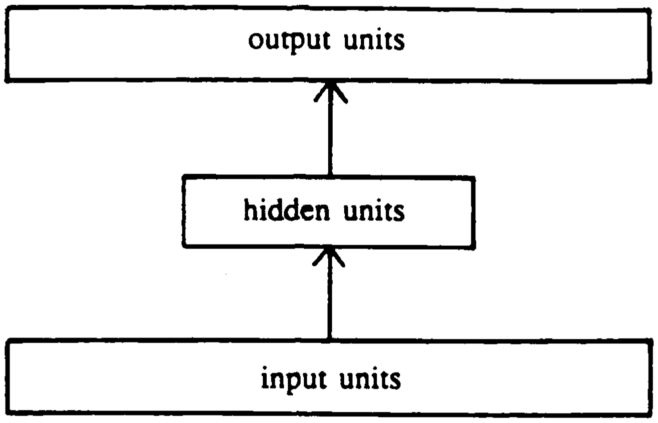


Fig. 1. A diagram of a three layer encoder network that learns good codes using back-propagation. On the forward pass, activity flows from the input units in the bottom layer to the output units in the top layer. On the backward pass, error-derivatives flow from the top layer to the bottom layer.

Instead of using a separate group of units for the input and output we use the very same group of "visible" units, so the input vector is the initial state of this group and the output vector is the state after information has passed around the loop. The difference between the activity of a visible unit before and after sending activity around the loop is the derivative of the squared reconstruction error. So, if the visible units are linear, we can perform gradient descent in the squared error by changing each of a visible unit's incoming weights by an amount proportional to the product of this difference and the activity of the hidden unit from which the connection emanates. So learning the weights from the hidden units to the output units is simple. The harder problem is to learn the weights on connections coming into hidden units because there is no direct specification of the desired states of these units. Back-propagation solves this problem by back-propagating error-derivatives from the output units to generate error-derivatives for the hidden units. Recirculation solves the problem in a quite different way that is easier to implement but much harder to analyse.

THE RECIRCULATION PROCEDURE

We introduce the recirculation procedure by considering a very simple architecture in which there is just one group of hidden units. Each visible unit has a directed connection to every hidden unit, and each hidden unit has a directed connection to every visible unit. The total input received by a unit is

$$x_j = \sum_i y_i w_{ji} - \theta_j \tag{1}$$

where y_i is the state of the i^{th} unit, w_{ji} is the weight on the connection from the i^{th} to the j^{th} unit and θ_j is the threshold of the j^{th} unit. The threshold term can be eliminated by giving every unit an extra input connection whose activity level is fixed at 1. The weight on this special connection is the negative of the threshold, and it can be learned in just the same way as the other weights. This method of implementing thresholds will be assumed throughout the paper.

The functions relating inputs to outputs of visible and hidden units are smooth monotonic functions with bounded derivatives. For hidden units we use the logistic function:

$$y_j = \sigma(x_j) = \frac{1}{1+e^{-x_j}} \tag{2}$$

Other smooth monotonic functions would serve as well. For visible units, our mathematical analysis focuses on the linear case in which the output equals the total input, though in simulations we use the logistic function.

We have already given a verbal description of the learning rule for the hidden-to-visible connections. The weight, w_{ij}, from the j^{th} hidden unit to the i^{th} visible unit is changed as follows:

$$\Delta w_{ij} = \varepsilon y_j(1)\,[y_i(0) - y_i(2)] \tag{3}$$

where $y_i(0)$ is the state of the i^{th} visible unit at time 0 and $y_i(2)$ is its state at time 2 after activity has passed around the loop once. The rule for the visible-to-hidden connections is identical:

$$\Delta w_{ji} = \varepsilon y_i(2)\,[y_j(1) - y_j(3)] \tag{4}$$

where $y_j(1)$ is the state of the j^{th} hidden unit at time 1 (on the first pass around the loop) and $y_j(3)$ is its state at time 3 (on the second pass around the loop). Fig. 2 shows the network exploded in time.

In general, this rule for changing the visible-to-hidden connections does not perform steepest descent in the squared reconstruction error, so it behaves differently from back-propagation. This raises two issues: Under what conditions does it work, and under what conditions does it approximate steepest descent?

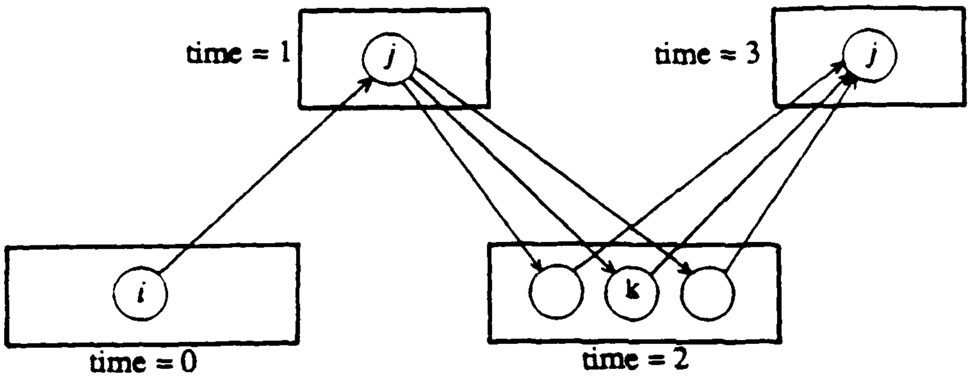


Fig. 2. A diagram showing the states of the visible and hidden units exploded in time. The visible units are at the bottom and the hidden units are at the top. Time goes from left to right.

CONDITIONS UNDER WHICH RECIRCULATION APPROXIMATES GRADIENT DESCENT

For the simple architecture shown in Fig. 2, the recirculation learning procedure changes the visible-to-hidden weights in the direction of steepest descent in the squared reconstruction error provided the following conditions hold:

1. The visible units are linear.
2. The weights are symmetrical (i.e. $w_{ji}=w_{ij}$ for all i,j).
3. The visible units have high regression.

"Regression" means that, after one pass around the loop, instead of setting the activity of a visible unit, i, to be equal to its current total input, $x_i(2)$, as determined by Eq 1, we set its activity to be

$$y_i(2) = \lambda y_i(0) + (1-\lambda)x_i(2) \tag{5}$$

where the regression, λ, is close to 1. Using high regression ensures that the visible units only change state slightly so that when the new visible vector is sent around the loop again on the second pass, it has very similar effects to the first pass. In order to make the learning rule for the hidden units as similar as possible to the rule for the visible units, we also use regression in computing the activity of the hidden units on the second pass

$$y_j(3) = \lambda y_j(1) + (1-\lambda)\sigma(x_j(3)) \tag{6}$$

For a given input vector, the squared reconstruction error, E, is

$$E = \frac{1}{2}\sum_k [y_k(2)-y_k(0)]^2$$

For a hidden unit, j,

$$\frac{\partial E}{\partial y_j(1)} = \sum_k \frac{\partial E}{\partial y_k(2)} \frac{dy_k(2)}{dx_k(2)} \frac{\partial x_k(2)}{\partial y_j(1)} = \sum_k [y_k(2) - y_k(0)]\, y_k'(2)\, w_{kj} \tag{7}$$

where

$$y_k'(2) = \frac{dy_k(2)}{dx_k(2)}$$

For a visible-to-hidden weight w_{ji}

$$\frac{\partial E}{\partial w_{ji}} = y_j'(1)\, y_i(0) \frac{\partial E}{\partial y_j(1)}$$

So, using Eq 7 and the assumption that $w_{kj} = w_{jk}$ for all k,j

$$\frac{\partial E}{\partial w_{ji}} = y_j'(1)\, y_i(0)\, [\sum_k y_k(2)\, y_k'(2)\, w_{jk} - \sum_k y_k(0)\, y_k'(2)\, w_{jk}]$$

The assumption that the visible units are linear (with a gradient of 1) means that for all k, $y_k'(2) = 1$. So using Eq 1 we have

$$\frac{\partial E}{\partial w_{ji}} = y_j'(1)\, y_i(0)\, [x_j(3) - x_j(1)] \tag{8}$$

Now, with sufficiently high regression, we can assume that the states of units only change slightly with time so that

$$y_j'(1)\, [x_j(3) - x_j(1)] \approx \sigma(x_j(3)) - \sigma(x_j(1)) = \frac{1}{(1-\lambda)} [y_j(3) - y_j(1)]$$

$$\text{and} \qquad y_i(0) \approx y_i(2)$$

So by substituting in Eq 8 we get

$$\frac{\partial E}{\partial w_{ji}} \approx \frac{1}{(1-\lambda)}\, y_i(2)\, [y_j(3) - y_j(1)] \tag{9}$$

An interesting property of Eq 9 is that it does not contain a term for the gradient of the input-output function of unit j so recirculation learning can be applied even when unit j uses an *unknown* non-linearity. To do back-propagation it is necessary to know the gradient of the non-linearity, but recirculation *measures* the gradient by measuring the effect of a small difference in input, so the term $y_j(3) - y_j(1)$ implicitly contains the gradient.

A SIMULATION OF RECIRCULATION

From a biological standpoint, the symmetry requirement that $w_{ij}=w_{ji}$ is unrealistic unless it can be shown that this symmetry of the weights can be learned. To investigate what would happen if symmetry was not enforced (and if the visible units used the same non-linearity as the hidden units), we applied the recirculation learning procedure to a network with 4 visible units and 2 hidden units. The visible vectors were 1000, 0100, 0010 and 0001, so the 2 hidden units had to learn 4 different codes to represent these four visible vectors. All the weights and biases in the network were started at small random values uniformly distributed in the range -0.5 to $+0.5$. We used regression in the hidden units, even though this is not strictly necessary, but we ignored the term $1/(1-\lambda)$ in Eq 9.

Using an ε of 20 and a λ of 0.75 for both the visible and the hidden units, the network learned to produce a reconstruction error of less than 0.1 on every unit in an average of 48 weight updates (with a maximum of 202 in 100 simulations). Each weight update was performed after trying all four training cases and the change was the sum of the four changes prescribed by Eq 3 or 4 as appropriate. The final reconstruction error was measured using a regression of 0, even though high regression was used during the learning. The learning speed is comparable with back-propagation, though a precise comparison is hard because the optimal values of ε are different in the two cases. Also, the fact that we ignored the term $1/(1-\lambda)$ when modifying the visible-to-hidden weights means that recirculation tends to change the visible-to-hidden weights more slowly than the hidden-to-visible weights, and this would also help back-propagation.

It is not immediately obvious why the recirculation learning procedure works when the weights are not constrained to be symmetrical, so we compared the weight changes prescribed by the recirculation procedure with the weight changes that would cause steepest descent in the sum squared reconstruction error (i.e. the weight changes prescribed by back-propagation). As expected, recirculation and back-propagation agree on the weight changes for the hidden-to-visible connections, even though the gradient of the logistic function is not taken into account in weight adjustments under recirculation. (Conrad Galland has observed that this agreement is only slightly affected by using visible units that have the non-linear input-output function shown in Eq 2 because at any stage of the learning, all the visible units tend to have similar slopes for their input-output functions, so the non-linearity scales all the weight changes by approximately the same amount.)

For the visible-to-hidden connections, recirculation initially prescribes weight changes that are only randomly related to the direction of steepest descent, so these changes do not help to improve the performance of the system. As the learning proceeds, however, these changes come to agree with the direction of steepest descent. The crucial observation is that this agreement occurs *after* the hidden-to-visible weights have changed in such a way that they are approximately aligned (symmetrical up to a constant factor) with the visible-to-hidden weights. So it appears that changing the hidden-to-visible weights in the direction of steepest descent creates the conditions that are necessary for the recirculation procedure to cause changes in the visible-to-hidden weights that follow the direction of steepest descent.

It is not hard to see why this happens if we start with random, zero-mean visible-to-hidden weights. If the visible-to-hidden weight w_{ji} is positive, hidden unit j will tend to have a higher than average activity level when the i^{th} visible unit has a higher than average activity. So y_j will tend to be higher than average when the reconstructed value of y_i should be higher than average -- i.e. when the term $[y_i(0)-y_i(2)]$ in Eq 3 is positive. It will also be lower than average when this term is negative. These relationships will be reversed if w_{ji} is negative, so w_{ij} will grow faster when w_{ji} is positive than it will when w_{ji} is negative. Smolensky[4] presents a mathematical analysis that shows why a similar learning procedure creates symmetrical weights in a purely linear system. Williams[5] also analyses a related learning rule for linear systems which he calls the "symmetric error correction" procedure and he shows that it performs principle components analysis. In our simulations of recirculation, the visible-to-hidden weights become aligned with the corresponding hidden-to-visible weights, though the hidden-to-visible weights are generally of larger magnitude.

A PICTURE OF RECIRCULATION

To gain more insight into the conditions under which recirculation learning produces the appropriate changes in the visible-to-hidden weights, we introduce the pictorial representation shown in Fig. 3. The initial visible vector, A, is mapped into the reconstructed vector, C, so the error vector is AC. Using high regression, the visible vector that is sent around the loop on the second pass is P, where the difference vector AP is a small fraction of the error vector AC. If the regression is sufficiently high and all the non-linearities in the system have bounded derivatives and the weights have bounded magnitudes, the difference vectors AP, BQ, and CR will be very small and we can assume that, to first order, the system behaves linearly in these difference vectors. If, for example, we moved P so as to double the length of AP we would also double the length of BQ and CR.

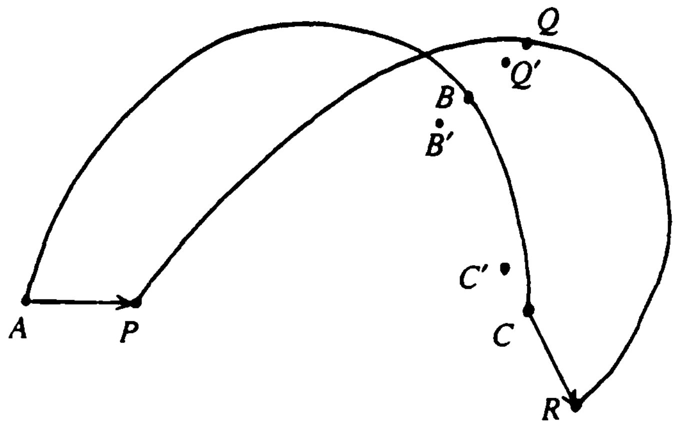


Fig. 3. A diagram showing some vectors (A, P) over the visible units, their "hidden" images (B, Q) over the hidden units, and their "visible" images (C, R) over the visible units. The vectors B' and C' are the hidden and visible images of A after the visible-to-hidden weights have been changed by the learning procedure.

Suppose we change the visible-to-hidden weights in the manner prescribed by Eq 4, using a very small value of ε. Let Q' be the hidden image of P (i.e. the image of P in the hidden units) after the weight changes. To first order, Q' will lie between B and Q on the line BQ. This follows from the observation that Eq 4 has the effect of moving each $y_j(3)$ towards $y_j(1)$ by an amount proportional to their difference. Since B is close to Q, a weight change that moves the hidden image of P from Q to Q' will move the hidden image of A from B to B', where B' lies on the extension of the line BQ as shown in Fig. 3. If the hidden-to-visible weights are not changed, the visible image of A will move from C to C', where C' lies on the extension of the line CR as shown in Fig. 3. So the visible-to-hidden weight changes will reduce the squared reconstruction error provided the vector CR is approximately parallel to the vector AP.

But why should we expect the vector CR to be aligned with the vector AP? In general we should not, except when the visible-to-hidden and hidden-to-visible weights are approximately aligned. The learning in the hidden-to-visible connections has a tendency to cause this alignment. In addition, it is easy to modify the recirculation learning procedure so as to increase the tendency for the learning in the hidden-to-visible connections to cause alignment. Eq 3 has the effect of moving the visible image of A closer to A by an amount proportional to the magnitude of the error vector AC. If we apply the same rule on the next pass around the loop, we move the visible image of P closer to P by an amount proportional to the magnitude of PR. If the vector CR is anti-aligned with the vector AP, the magnitude of AC will exceed the magnitude of PR, so the result of these two movements will be to improve the alignment between AP and CR. We have not yet tested this modified procedure through simulations, however.

This is only an informal argument and much work remains to be done in establishing the precise conditions under which the recirculation learning procedure approximates steepest descent. The informal argument applies equally well to systems that contain longer loops which have several groups of hidden units arranged in series. At each stage in the loop, the same learning procedure can be applied, and the weight changes will approximate gradient descent provided the difference of the two visible vectors that are sent around the loop aligns with the difference of their images. We have not yet done enough simulations to develop a clear picture of the conditions under which the changes in the hidden-to-visible weights produce the required alignment.

USING A HIERARCHY OF CLOSED LOOPS

Instead of using a single loop that contains many hidden layers in series, it is possible to use a more modular system. Each module consists of one "visible" group and one "hidden" group connected in a closed loop, but the visible group for one module is actually composed of the hidden groups of several lower level modules, as shown in Fig. 4. Since the same learning rule is used for both visible and hidden units, there is no problem in applying it to systems in which some units are the visible units of one module and the hidden units of another. Ballard[6] has experimented with back-propagation in this kind of system, and we have run some simulations of recirculation using the architecture shown in Fig. 4. The network learned to encode a set of vectors specified over the bottom layer. After learning. each of the vectors became an attractor and the network was capable of completing a partial vector, even though this involved passing information through several layers.

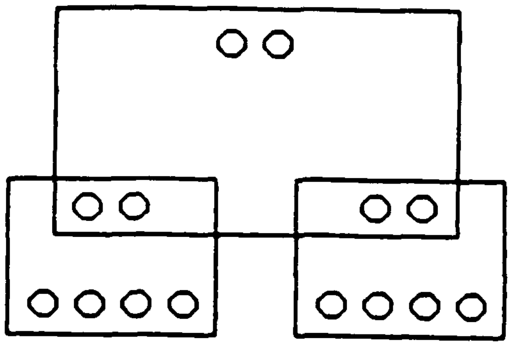


Fig 4. A network in which the hidden units of the bottom two modules are the visible units of the top module.

CONCLUSION

We have described a simple learning procedure that is capable of forming representations in non-linear hidden units whose input-output functions have bounded derivatives. The procedure is easy to implement in hardware, even if the non-linearity is unknown. Given some strong assumptions, the procedure performs gradient descent in the reconstruction error. If the symmetry assumption is violated, the learning procedure still works because the changes in the hidden-to-visible weights produce symmetry. If the assumption about the linearity of the visible units is violated, the procedure still works in the cases we have simulated. For the general case of a loop with many non-linear stages, we have an informal picture of a condition that must hold for the procedure to approximate gradient descent, but we do not have a formal analysis, and we do not have sufficient experience with simulations to give an empirical description of the general conditions under which the learning procedure works.

REFERENCES

1. D. E. Rumelhart, G. E. Hinton and R. J. Williams, *Nature* **323**, 533-536 (1986).

2. D. H. Ackley, G. E. Hinton and T. J. Sejnowski, *Cognitive Science* **9**,147-169 (1985).

3. G. Cottrell, J. L. Elman and D. Zipser, Proc. Cognitive Science Society, Seattle, WA (1987).

4. P. Smolensky, Technical Report CU-CS-355-87, University of Colorado at Boulder (1986).

5. R. J. Williams, Technical Report 8501, Institute of Cognitive Science, University of California, San Diego (1985).

6. D. H. Ballard, Proc. American Association for Artificial Intelligence, Seattle, WA (1987).

Submitted to the 1988 Cognitive Science Conference

Applying Contextual Constraints in Sentence Comprehension

Mark F. St. John and James L. McClelland

Carnegie-Mellon University

The goal of our research has been to develop a model that converts a simple sentence into a conceptual representation of the event that the sentence describes: specifically, a model that converts the constituent phrases of a sentence into a representation of an event, that assigns a thematic role to each constituent (Fillmore, 1968), and that interprets ambiguous and vague words. In our model, the comprehension process is viewed as a form of constraint satisfaction. The surface features of a sentence, its particular words and their order and morphology provide a rich set of constraints on the sentence's meaning. These constraints vary in strength and compete or cooperate according to their strength to produce an interpretation of the sentence.

Determining the exact constraints, and their appropriate strengths, is difficult, but a connectionist learning procedure allows a model to learn them. The learning procedures take a statistical approach to the task. By comparing large numbers of sentences to the events they describe, the many-to-many mapping, between features of the sentences and events, emerges as statistical regularities.

Often a sentence will omit information about the event it describes. We wanted our model to infer this missing information: to represent the event as completely as possible. Sometimes, though, a sentence is compatible with more than one interpretation. In "The private shot the target," the instrument is sometimes a rifle, and sometimes a pistol. In such situations, a good long-term strategy is to represent each possibility according to its likelihood in the given context.

As each constituent of a sentence is processed, the context changes. The processor should update its inferences to reflect the changing context. It should also adjust its interpretation of previous material. The model should utilize information derived from the sentence immediately (Carpenter & Just, 1977; Marslen-Wilson & Tyler, 1980). In all, therefore, we have six goals for our model of sentence comprehension:

* to disambiguate ambiguous words
* to instantiate vague words
* to assign thematic roles
* to elaborate implied roles
* to learn to perform these tasks
* to immediately adjust its interpretation

Overview of the Model

Processing

A sentence is represented as a sequence of phrases and each is processed in turn. The information each phrase provides is immediately used to update a representation of the event. This representation is called the sentence gestalt because all of the information from the sentence is represented together within a single, distributed representation. The event described by a sentence is represented as a pattern of activity across the units of this representation.

To process the constituent phrases of a sentence, we adapted an architecture from Jordan (1986) that uses the output of previous processing as input on the next iteration. To process a phrase, the appropriate *phrase* units are activated and the *sentence gestalt* activations (initially zero) are copied over to the *previous sentence gestalt* units. Activation from these layers combine in a hidden layer and create a new pattern over the *sentence gestalt* units (see Figure 1). Each phrase of the sentence is processed in sequence.

Though other models have used a type of sentence gestalt to represent the meaning of a sentence (McClelland & Kawamoto, 1986; St. John & McClelland, 1987), ours is the first to make the gestalt a trainable layer of hidden units. The advantage is that the network can optimize its representation to include only information that is relevant to its task. Since a layer of hidden units cannot be trained directly, we invented a way of "decoding" the representation into an output layer. The output layer represents the event as a set of thematic role and filler pairs. For example, the event described by "The pitcher

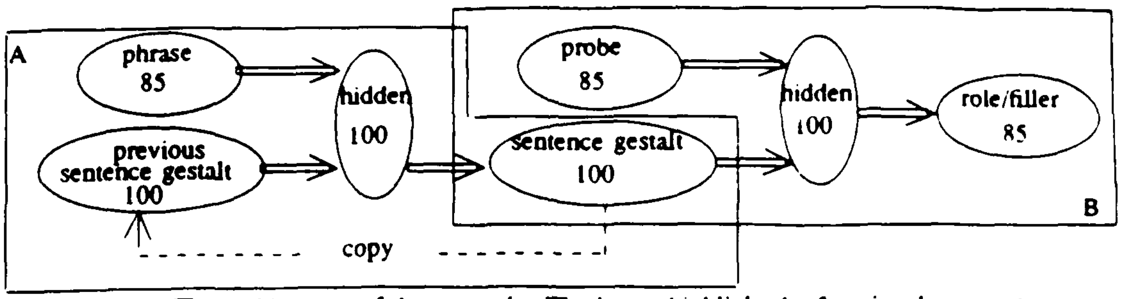


Figure 1. The architecture of the network. The boxes highlight the functional parts: Area A processes the phrases into the sentence gestalt, and Area B processes the sentence gestalt into the output representation.

threw the ball" would be represented as the set {(agent, pitcher/ball-player), (action, threw/toss), (patient, ball/sphere)}.

The output layer can represent one role/filler pair at a time. To decode a particular role/filler pair, the sentence gestalt is probed with half of the pair. Activation from the probe and the *sentence gestalt* combine in another hidden layer which then activates the entire pair in the output layer. The entire event can be decoded in this way by successively probing with each half of each pair.

When more than one object can plausibly fill a role, the model maximizes its long-term success by activating each filler to the degree it is likely. More formally, the activation of each filler should correspond to its conditional probability of occurring in the given context. The network should learn weights to produce these activations through training. To achieve this goal, we employed an error measure in the learning procedure, cross-entropy (Hinton, 1987), that converges on this goal:

$$C = -\sum_j [T_j \log_2 (A_j) + (1-T) \log_2 (1-A_j)]$$

where T_j is the target activation and A_j is the output activation of unit j. As with any connectionist learning procedure, the goal is to minimize the error measure or cost-function (cf. Hinton, 1987). When C is minimized across all the sentences of the training corpus, the activation of a particular output unit is equal to the conditional probability that whatever the unit represents is true given the current situation. Specifically, if each unit represented the occurrence of a particular filler in an event, that unit's activation would represent the conditional probability of that filler occurring given what was currently known about the event. In minimizing C, the network is searching for weights that allow it to match activations to conditional probabilities.

Environment and training

Training consists of trials in which the network is presented with a sentence and the event it describes. The network is trained to generate the event from the sentence as input. These sentence/event pairs were generated on-line for each training trial. Some pairs were more likely to be generated than others. Over training, these likelihood differences translated into differences in training frequency and created regularities.

To promote immediate processing a special training regime is used. After each phrase has been processed, the network is trained to predict the set of role/filler pairs of the entire sentence. From the first phrase of the sentence, then, the model is forced to try to predict the entire event. This training procedure forces the model to do as much immediate processing as possible. Consequently, as each new phrase is processed, the model's predictions of the event are refined to reflect the additional evidence it supplies.

An illustration of processing

As the network processes "The adult ate the steak," the *sentence gestalt* can be probed to see what it is representing after each phrase is processed (see Figure 2). After processing "the adult," the gestalt represents that the agent of the event is an adult, and it guesses weakly at a number of actions. Following "ate," the network encodes that information and expects the patient to be food. Once "the steak" is processed, the network represents steak as the patient and infers that knife is the instrument. It also is able to revise its representation of the agent. Because one person, the busdriver, is the most frequent steak-eater in the corpus, the network infers that the adult from the sentence is the busdriver. In this way, the model infers missing thematic roles and immediately adjusts previous results (in this case by instantiating the agent) as more information becomes available.

Specifics of the Model

Input representation

Each phrase was encoded by a single unit representing the noun or verb. No semantic information was provided. For prepositional phrases, the preposition was encoded by a second unit. The passive voice was encoded by a unit in the verb phrase representation. One unit stood for each of 13 verbs, 31 nouns, 4 prepositions, 3 adverbs, and 7 ambiguous words. Two of the ambiguous words had two verb meanings, three had two noun meanings, and two had a verb and a noun meaning. Six of the words were vague terms (e.g. someone, something, and food).

The relative location of each phrase within the sentence was also encoded in the input. It was coded by four units that represent location respective to the verb: pre-verbal, verbal, first-post-verbal, and n-post-verbal. The first-post-verbal unit was active for the phrase immediately following the verb, and the n-post-verbal unit was active for any phrase

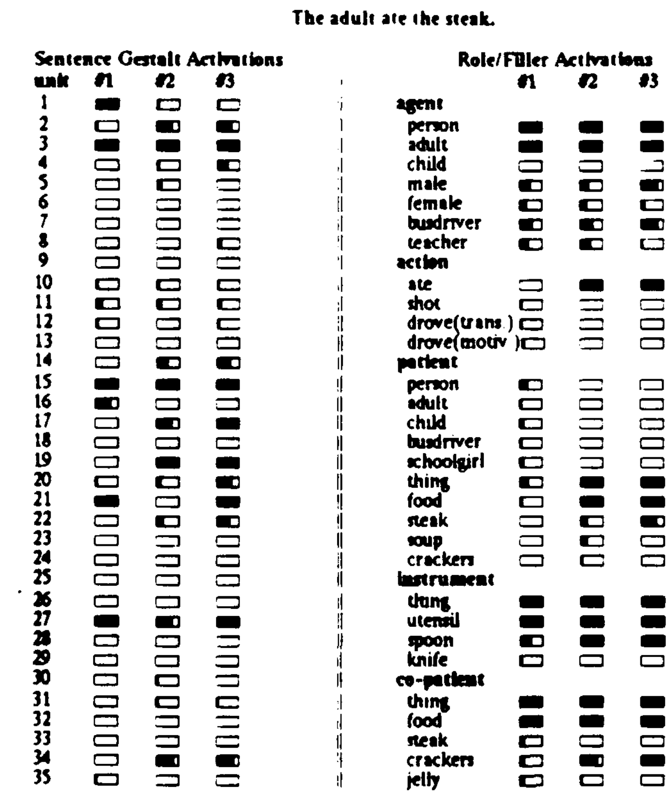


Figure 2. The evolution of the sentence gestalt during processing of a sentence. The # corresponds to the number of phrases that have been presented to the network at that point. #1 means the network has seen the first phrase; #2 means it has seen the first two phrases; etc. The activations (ranging between 0 and 1) of a sampling of units are graphed as the darkened area of each box. By probing the gestalt with the role half of each role/filler pair, the event represented by each gestalt can be observed. The role probe and the activation level of active filler units in the output layer are presented for each gestalt.

occurring after the first-post-verbal phrase. A number of phrases, therefore, could share the n-post-verbal position. For example, "The ball was hit by someone with the bat in the park," was encoded as the ordered set {(pre-verbal, ball), (verbal, passive-voice, hit), (first-post-verbal, by, someone), (n-post-verbal, with, bat), (n-post-verbal, in, park)}.

Output representation

The output had one unit for each of 9 possible thematic roles (e.g. agent, action, patient, instrument) and one unit for each of 28 concepts, 14 actions, and 3 adverbs. Additionally, 13 "feature" units, like male, female, and adult, were included in the output to allow the demonstration of more subtle effects of constraints on interpretation. Any one role/filler pattern, then, consisted of two parts: for the role, one of the 9 role units was active, and for the filler, a unit representing the concept, action, or adverb was active. If relevant, any of the feature units or the passive voice unit were active.[1] This representation is not meant to be comprehensive. Instead, it is meant to be a convenient way to train and demonstrate the processing abilities of the network.[2]

Training environment

While the sentences often include ambiguous or vague words, the events are always specific and complete: each thematic role related to an action is filled by some specific concept. Accordingly, each action occurs in a particular location, and actions requiring an instrument always have a specific instrument. The event would be generated by picking concepts to fill each thematic role related to the event according to preset probabilities. The probability of selecting each concept depended upon what else had been selected for that event. Conversely, the sentences often omit thematic roles (e.g. instrument and location) from mention and use vague words (e.g. someone and something) to refer to parts of an event.

The sentences had to be limited in complexity because of limitations in the event representation. Only one filler could be assigned to a role in a particular sentence. Also, all the roles are assumed to belong to the sentence as a whole. Therefore, no embedded clauses or phrases attached to single constituents were possible.

On each training trial, the error, in terms of cross-entropy, was propagated backward through the network (cf. Rumelhart, Hinton, & Williams, 1986). The weight changes from each trial were added together and used to update the weights every 60 trials. This consolidation of weight changes tends to smooth out the aberrations caused by the random generation of training examples.

Performance

Processing in general

The simulation was stopped and evaluated after 330,000 sentence trials. First, we will assess the model's ability to comprehend sentences generally. Then we will examine the model's ability to fulfill our specific processing goals. One hundred sentences were drawn randomly from the corpus. The probability of drawing a sentence was the same as during training. Consequently, frequently practiced sentences were more common among the 100 test sentences than infrequently practiced sentences. Thirteen of these sentences were completely ambiguous. For example, the sentence, "The adult drank the iced-tea," can be instantiated with either busdriver or teacher as the agent, but the sentence offers no clues that busdriver is the correct agent in this particular sentence/event pair in the test set.

[1]If the word related to the concept appeared in a prepositional phrase, such as "with the knife," the appropriate preposition unit was also activated.

[2]A second output layer was included in the simulations. This layer reproduced the input phrase that fit with the role/filler pair being probed. Consequently, the model was required to retain the specific words in the sentence as well as their meaning. Since this aspect of the processing does not fit into the context of the current discussion, these units are not discussed further.

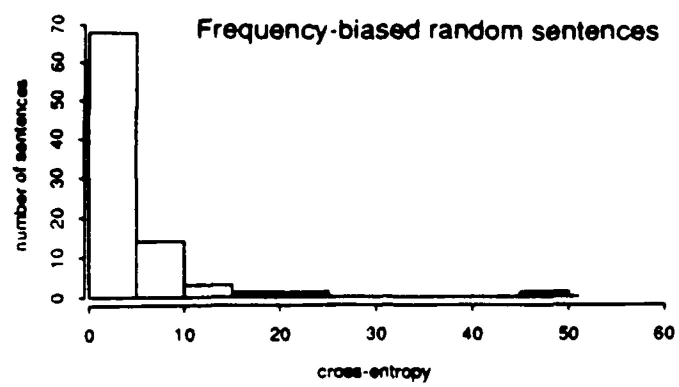


Figure 3. Histogram of the cross-entropy for frequency-biased random sentences. Sentences that were processed almost perfectly produced a small cross-entropy measure of between 0 and 10: only small errors occurred when an output unit should have been completely activate (with a value of 1), but only obtained an activation of .7 or .8, or when a unit should have had an activation of 0, but had an activation of .1 or .2. The incorrect activation of one role/filler pair produced a larger cross-entropy: between 15 and 20. For example, if teacher were supposed to be the agent, but the network activated busdriver, an error of about 15 would result.

Since the network cannot predict the event, these sentences were not tested. The remaining sentences were tested, and figure 3 presents a histogram of the results. Nearly every sentence was processed correctly.

Performance on specific processes

Our specific interest was to develop a processor that could correctly perform several important language comprehension processes. Five typical sentences were drawn from the corpus to test each processing task. The role assignment category was divided into groups according to the primary type of information used. Sentences in the active-voice semantic group contain semantic information relevant to assigning roles. In the example in Table 1, the network assigns schoolgirl to the role of agent. Of the concepts referred to in the sentence, only the schoolgirl has features which match an agent of stirring. Similarly, semantic knowledge constraints kool-aid to be the patient. Sentences in the passive-voice semantic category work similarly. The model processed each test sentence correctly.

To process sentences in the active and passive word order categories, however, the network cannot rely entirely on semantic constraints to assign thematic roles. In the sentence, "The busdriver was kissed by the teacher," the busdriver is as likely to be the

Processing tasks

Category	**Example**
Role assignment	
Active semantic	The schoolgirl stirred the kool-aid with a spoon.
Passive semantic	The ball was hit by the pitcher.
Active word order	The busdriver gave the rose to the teacher.
Passive word order	The busdriver was kissed by the teacher.
Word ambiguity	The pitcher hit the bat with the bat.
Concept instantiation	The teacher kissed someone.
Role elaboration	The teacher ate the soup (with a spoon).

Table 1. The four categories of processing tasks and an example sentence of each. Role assignment was tested under four conditions to assess the use of both semantic and word order information. The remaining three categories involve inferences about the content of a sentence. The parentheses in the role elaboration example denote a role that is not presented in the sentence, and must be inferred from the context.

agent as the patient. Only the relative location information, in conjunction with the passive cues, can cue the correct role assignments. In active sentences, the pre-verbal phrase is the agent and the post-verbal phrase is the patient. In passive sentences, on the other hand, the pre-verbal phrase is the patient and the post-verbal phrase is the agent. The model processed each test sentence correctly.

The remaining three categories involve the use of context to further specify the concepts referred to in a sentence. Sentences in the word ambiguity category contain ambiguous words. While the word itself cues two different interpretations, the context fits only one. In "The pitcher hit the bat with the bat," pitcher cues both container and ball player. The context cues both ball player and busdriver because the model has seen sentences involving both people hitting bats. All the constraints supporting ball player combine, and together they win the competition for the interpretation of the sentence. Even when several words of a sentence are ambiguous, the event which they support in common dominates the disparate events that they support individually. Consequently, the final interpretation of each word fits together into a globally consistent event. For each test sentence, even when several words were ambiguous, the model performed correctly.

Concept instantiation works similarly. Though the word cues a number of more specific concepts, only one fits the context. Again, the constraints from the word and from the context combine to produce a unique, specific interpretation of the term. Depending upon the sentence, however, the context may only partially constrain the interpretation. Such is the case in "The teacher kissed someone." "Someone" could refer to any of the four people found in the corpus. Since, in the network's experience, females only kiss males, the context constrains the interpretation of "someone" to be either the busdriver or the pitcher, but no further. Consequently, the model activates the male and person features of the patient while leaving the units representing busdriver and pitcher partially and equally active. In general, the model is capable of inferring as much information as the evidence permits: the more evidence, the more specific the inference.

Finally, sentences in the role elaboration category test the model's ability to infer thematic roles not mentioned in the input sentence. For example, in "The teacher ate the soup," no instrument is mentioned, yet a spoon can be inferred. Here, the context alone provides the constraints for making the inference. Extra roles that are very likely are inferred strongly. When the roles are less likely, or when more than one concept can fill a role, the concepts are only weak inferred. Again, the model processed each test sentence correctly.

Conclusions

Parallel Distributed Processing models have a number of qualities that make them useful as models of language comprehension. First, they allow the simultaneous processing of many constraints. Each bit of relevant information can be applied to a computation. This feature allows far greater interaction between constraints in computing an interpretation of a sentence. As Marcus (1980) points out, the competition and cooperation among constraints of varying strength is an essential aspect of comprehension. Second, PDP models allow the strength of constraints to vary on a continuum rather than discretely. Information may be more or less reliable, and correlations may be more or less strong. The activation level of a unit allows the quality of the information to be represented explicitly and combined effectively. This feature allows *quantitative* interaction among constraints. Third, PDP models naturally perform pattern completion. Given some input, the models add default information to produce an elaborated representation. Importantly, this default information is tailored to the specific input presented. This feature allows context specific inferences about the input to be computed.

Because PDP models learn, useful and complex constraints develop of their own accord. This learning process is slow, requiring a great deal of practice. While comprehension involving strong and regular constraints are learned relatively rapidly, irregular and complex constraints are only learned very slowly. Early on, therefore, the network begins to perform passably, and it slowly improves to correctly process more sentences.

The model handles ambiguity robustly by processing the input in terms of constraints on a conceptual representation of an event. Within this framework, word disambiguation and concept instantiation are similar processes. Both ambiguous words and vague terms provide conflicting constraints on their interpretation. The constraints from the word itself and additional constraints from its context provide evidence, respectively, for the general and specific features of the concept referred to. Role elaboration is similar, but at the level of interpreting the sentence as a whole. The features of the sentence provide constraints on the unspecified features of the event, such as implicit thematic roles like location, instrument, and manner. When more than one role/filler is likely, each is represented according to its conditional probability.

The interpretation of a sentence evolves as each word is presented to the network. The model sequentially processes each word and immediately adjusts its interpretation of old information and creates expectations of additional information as it attempts to represent the entire conceptual event.

The model cannot, however, represent sentences with embedded clauses. Extending the model to represent more complex sentences is an important goal. Meanwhile, we have learned a great deal about viewing sentence comprehension as a process of weak constraint satisfaction.

References

Carpenter, P. A. & Just, M. A. (1977). Reading comprehension as the eyes see it. In M. A. Just & P. A. Carpenter (Eds.), *Cognitive processes in comprehension*. Hillsdale, NJ: Erlbaum.

Fillmore, C. J. (1968). The case for case. In E. Bach & R. T. Harms (Eds.), *Universals in linguistic theory*. New York: Holt, Rinehart, & Winston.

Hinton, G. E. (1987). Connectionist learning procedures. Tech report #CMU-CS-87-115.

Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. Paper presented to the 8th Annual Conference of the Cognitive Science Society. Amherst, MA.

MacWhinney, B. (1987). Competition. In B. MacWhinney (Ed.), *Mechanisms of language acquisition: The 20th annual Carnegie symposium on cognition.* Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers.

Marcus, M. P. (1980). *A theory of syntactic recognition for natural language*. Cambridge, MA: MIT Press.

Marslen-Wilson, W. & Tyler, L. K. (1980). The temporal structure of spoken language understanding. *Cognition, 8*, 1-71.

McClelland, J. L. & Kawamoto, A. H. (1986). Mechanisms of sentence processing: Assigning roles to constituents. In J. L. McClelland, D. E. Rumelhart, and the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 2*. Cambridge, MA.: MIT Press.

Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1*. Cambridge, MA.: MIT Press.

St. John, M. F. & McClelland, J. L. (1987). Reconstructive memory for sentences: A PDP approach. Paper presented to the Ohio University Inference Conference, *Proceedings Inference: OUIC 86*. University of Ohio, Athens, OH.

Using Fast Weights to Deblur Old Memories

Geoffrey E. Hinton and David C. Plaut

Computer Science Department
Carnegie-Mellon University

Abstract

Connectionist models usually have a single weight on each connection. Some interesting new properties emerge if each connection has two weights: A slowly changing, plastic weight which stores long-term knowledge and a fast-changing, elastic weight which stores temporary knowledge and spontaneously decays towards zero. If a network learns a set of associations and then these associations are "blurred" by subsequent learning, *all* the original associations can be "deblurred" by rehearsing on just a few of them. The rehearsal allows the fast weights to take on values that temporarily cancel out the changes in the slow weights caused by the subsequent learning.

1. Introduction

Most connectionist models have assumed that each connection has a single weight which is adjusted during the course of learning. Despite the emerging biological evidence that changes in synaptic efficacy at a single synapse occur at many different time-scales (Kupferman, 1979; Hartzell, 1981), there have been relatively few attempts to investigate the computational advantages of giving each connection several different weights that change at different speeds. Even for phenomena like short-term memory where fast-changing weights might seem appropriate, connectionist models have typically used the activation levels of units rather than the weights to store temporary memories (Little and Shaw, 1975; Touretzky and Hinton, 1985). We know very little about the range of potential uses of fast weights. How do they alter the way networks behave, and what extra computational properties do they provide?

In this paper we assume that each connection has both a fast, elastic weight and a slow, plastic weight. The slow weights are like the weights normally used in connectionist networks--they change slowly and they hold all the long-term knowledge of the network. The fast weights change more rapidly and they continually regress towards zero so that their magnitude is determined solely by their recent past. The effective weight on the connection is the sum of these two.

At any instant, we can think of the system's knowledge as consisting of the slow weights with a temporary overlay of fast weights. The overlay gives a temporary context--a temporary associative memory that allows networks to do more flexible information processing. Many ways of using this temporary memory have been suggested.

1. It can be used for rapid temporary learning. When presented with a new association the network can store it in one trial, provided the storage only needs to be temporary.
2. It can be used for creating temporary bindings between features. Recent work by Von der Malsburg (1981) and Feldman (1982) has shown that fast-changing weights can be used to dynamically bind together a number of different properties into a coherent whole, or to discover approximate homomorphisms between two structured domains.
3. It can be used to allow truly recursive processing. During execution of a procedure, the

values of local variables and the stage reached in the procedure (the program counter) can be stored in the fast weights. This allows the procedure to call a subprocedure whose execution involves different patterns of activity in the very same units as the calling procedure. When the subprocedure has finished using the units, the state of the calling procedure can be reconstructed from the temporary associative memory. In this way the state does not need to be stored in the *activity* of the units, so the very same units can be used for running the subprocedure. A working simulation of this kind is described briefly in McClelland & Kawamoto (1986).

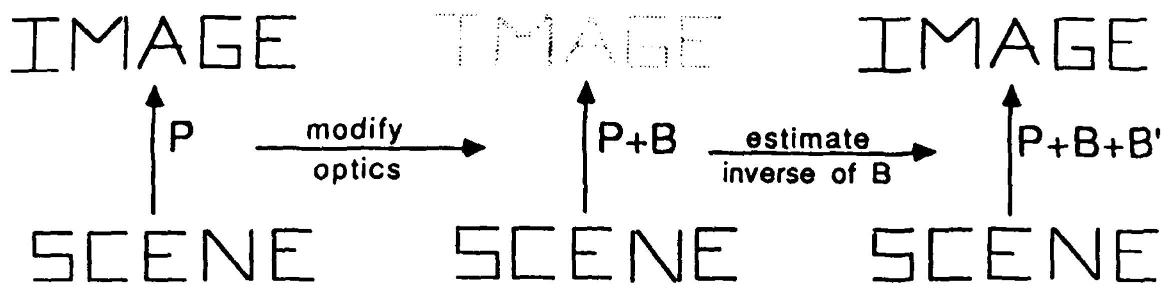
4. It can be used to implement a learning method called "shortest descent" which is a way of minimizing the amount of interference caused by new learning. Shortest descent will be described in a separate paper and is not discussed further here.

In this paper we describe a novel use for a temporary memory stored in the fast weights: it can be used for cancelling out the interference in a set of old associations caused by more recent learning. Consider a network which has slowly and painfully learned a set of associations in its slow weights. If this network is then taught a new set of associations without any more rehearsal of the old associations, there is normally some degradation of the old associations. We show that by using fast weights it is possible to quickly restore a whole set of old associations by rehearsing on just a subset of them. The fast weights cancel out the changes in the slow weights that have occurred since the old associations were learned, so the combination of the current slow weights and the fast weights approximates the earlier slow weights. The fast weights therefore create a context in which the old associations are present again. When the fast weights decay away, the new associations return.

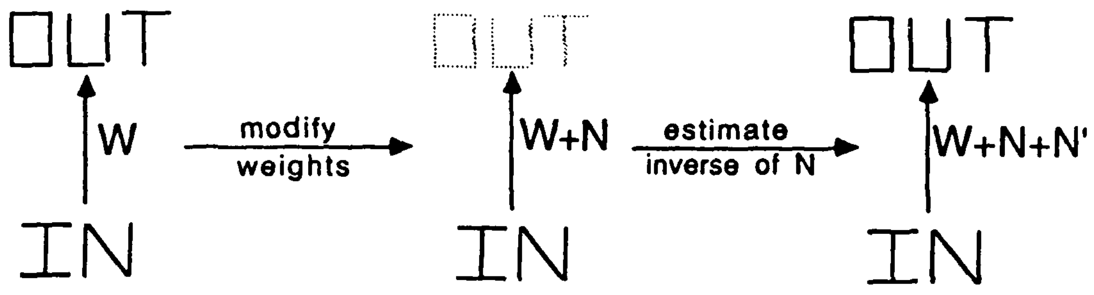
2. A deblurring analogy

There is an analogy between the use of fast weights to recover unretrained associations and a technique called "deblurring" which is sometimes used for cleaning up blurred images. Suppose that you are in your office and you want to take some photographs of what is on your computer screen. You set up a tripod in front of the screen and then you carefully focus the camera and take one photograph. Unfortunately, before you can take any more, your office mate moves the camera to a tripod in front of his screen and refocuses it. You now move the camera back to your tripod and it seems as if you must refocus, but there is an interesting alternative. You take another photograph of the same screen without refocussing and you compare it with your first photograph. The first photo is the "desired output" of the process that maps from screens to photos, and the difference between it and the "actual output" achieved with the out-of-focus camera can be used to estimate a deblurring operator which can be applied to the actual output to convert it to the desired output. This same deblurring operator can then be applied to any image taken with the out-of-focus camera. Focussing the camera is analogous to learning the slow weights that are required to map from input vectors (screens) to output vectors (photos). Estimating the deblurring operator is analogous to learning the fast weights that are required to compensate for the noise that has been added to the slow weights since the original learning (see figure 1).

The advantage of using fast weights rather than slow ones for deblurring is that it does not permanently interfere with the new associations. As soon as the fast weights have decayed back to zero, the new knowledge is restored.



(a)



(b)

Figure 1: An illustration of the deblurring analogy between (a) images and (b) networks. The laws of projection (P) determine the mapping from the scene to the image. After applying a blurring function (B), applying the inverse of B to the blurred image restores the image. Similarly, the weights (W) define a mapping from input to output. After noise (N) is added, determining and applying the inverse of N allows the network to produce the original output.

3. The learning procedure

We used the back propagation learning procedure (Rumelhart *et al.*, 1986a; 1986b) to investigate the properties of networks that learn with both fast and slow weights. We summarize the procedure below–the full mathematical details can be found in the above references.

The procedure operates on layered, feed-forward networks of deterministic, neuron-like units. These networks have a layer of input units at the bottom, any number of intermediate layers of hidden units, and a layer of output units at the top. Connections are allowed only from lower to higher layers.

The aim of the learning procedure is to find a set of weights on the connections such that, when the network is presented with each of a set of input vectors, the output vector produced by the network is sufficiently close to the corresponding desired output vector. The error produced by the network is defined to be the squared difference between the actual and desired output vectors summed over all input-output cases. The learning procedure minimizes this error by performing gradient descent in weight space. This requires adjusting each weight in the network in proportion to the partial derivative of the error with respect to that weight. These error derivatives are calculated in two passes.

The forward pass determines the output (or *state*) of each unit in the network. An input vector is presented to the network by setting the states of the input units. Layers are then processed sequentially, working from the bottom upward, with the states of units within a layer set in parallel. The input to a unit is the scalar product of the states of units in lower layers from which it receives connections, and the weights on these connections. The output of a unit is a real-valued smooth non-linear function of its input. The forward pass ends when the states of the output units are set.

The backward pass starts with the output units at the top of the network and works downward through each successive layer, "back-propagating" error derivatives to each weight in the network. For each layer, computing the error derivatives of the weights on incoming connections involves first computing the error derivatives with respect to the outputs, and then with respect to the inputs, of the units in that layer. The simple form of the unit input-output functions makes these computations straightforward. The backward pass is complete when the derivative of the error with respect to each weight in the network has been determined.

The simplest version of gradient descent is to decrement each weight in proportion, ε, to its error derivative. A more complicated version that usually converges faster is to add to the current weight change a proportion, α, of the previous weight change. This is analogous to introducing *momentum* to movement in weight space.

Since the effective weight on a connection is the sum of the fast and slow weights, these weights experience the same error derivative. Hence they behave differently only because they are modified using different weight change parameters ε and α, and because the fast weights decay towards zero. This is achieved by reducing the magnitude of each fast weight by some fraction, h, after each weight change.

4. A simulation of the deblurring effect

To demonstrate the deblurring effect, we used a simple task and a simple network. The task was to associate random binary 10-bit input vectors with random binary 10-bit output vectors. We selected 100 input vectors at random (without replacement) from the set of all 2^{10} binary vectors of length 10, and for each input vector we chose a random output vector. The network we used had three layers: 10 input units, 100 hidden units, and 10 output units. Each input unit was connected to all the hidden units, and each hidden unit was connected to all the output units. The hidden and output units also had variable biases that were modified during the learning.

We trained the network by repeatedly sweeping through the whole set of 100 associations and changing the weights after each sweep. After prolonged training (1300 sweeps with $\varepsilon = .02$ and $\alpha = .9$) the network knew the associations perfectly, and all the knowledge was in the slow weights. The fast weights were very close to zero because the errors were very small towards the end of the training, so the tiny error derivatives were dominated by the tendency of the fast weights to decay towards zero by 1% after each weight update.

Once the 100 associations were learned, we trained the ne'.work on 5 new random associations without further rehearsal on the original 100. Again, we continued the training until the new knowledge was in the slow weights (400 sweeps). We then retrained the network on only a subset of the original associations, and compared the improvement in performance on this retrained subset with the (incidental) improvement in performance on the rest of the associations. The main result, shown in figure 2, was that in the early stages of retraining the improvements in the associations that were not retrained were very nearly as good as in the associations that were explicitly retrained. This rather surprising result held even if only 10% of the old associations were retrained.

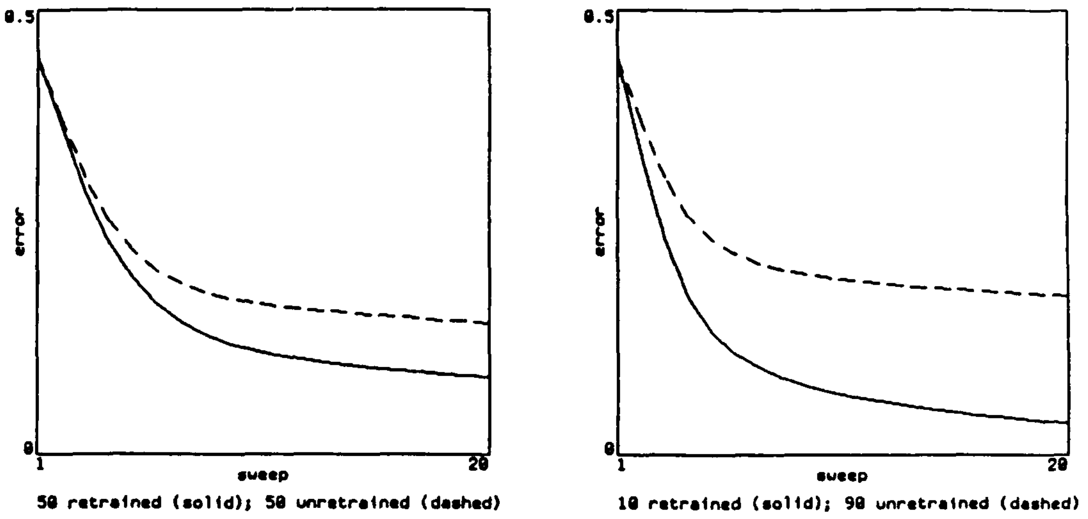


Figure 2: Performance on retrained and unretrained subsets of 100 input-output cases after (1) learning all 100 cases; and (2) interfering with the weights by learning 5 unrelated cases. The average error of an output unit is shown for the first 20 sweeps through the retrained subset.

The reason for this effect is that the knowledge about each association is distributed over many different connections, and when the network is retrained on *some* of the associations *all* the weights are pushed back towards the values that they used to have after the associations were first learned. So there is improvement in the unretrained associations even though they are only randomly related to the retrained ones. Of course, if the retrained and unretrained associations share some regularities the transfer will be even better.

4.1. A geometric explanation of the transfer effect

Consider the very simple network shown in figure 3. There are two input units which are directly connected to a single, linear output unit. The network is trained on two different associations each of which maps a two component input vector into a one component output vector. For each of the two input vectors, there will be many different combinations of the weights w_1 and w_2 that give the desired output vector, and since the output unit is linear, these combinations will form straight lines in weight space as shown is figure 4. In general, the only combination of weights that will satisfy both associations lies at the intersection of the two lines. So in this simple network, a gradient descent learning procedure will converge on the intersection point.

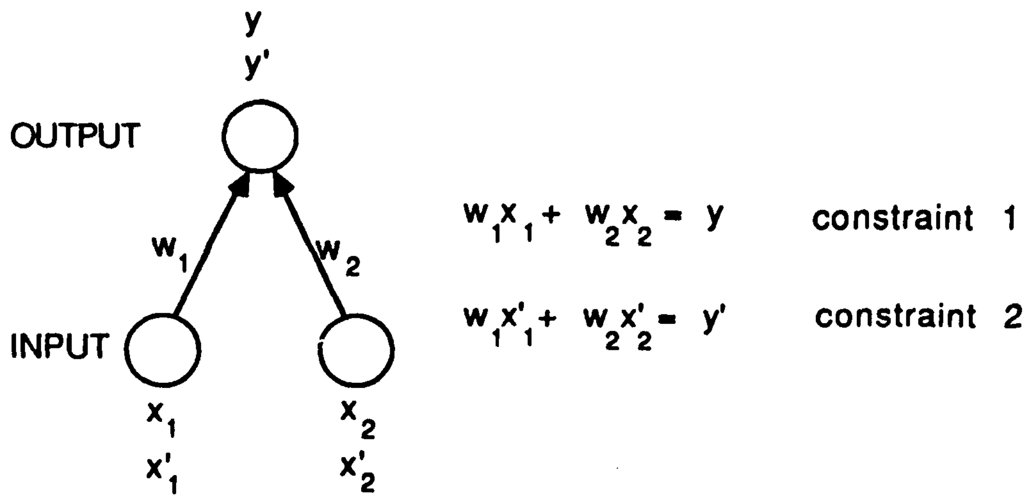

y
y'
OUTPUT
w_1
w_2
INPUT
x_1
x'_1
x_2
x'_2
$w_1x_1 + w_2x_2 = y$ constraint 1
$w_1x'_1 + w_2x'_2 = y'$ constraint 2

Figure 3: A simple network that learns to associate (x_1, x_2) with y and (x_1', x_2') with y'.

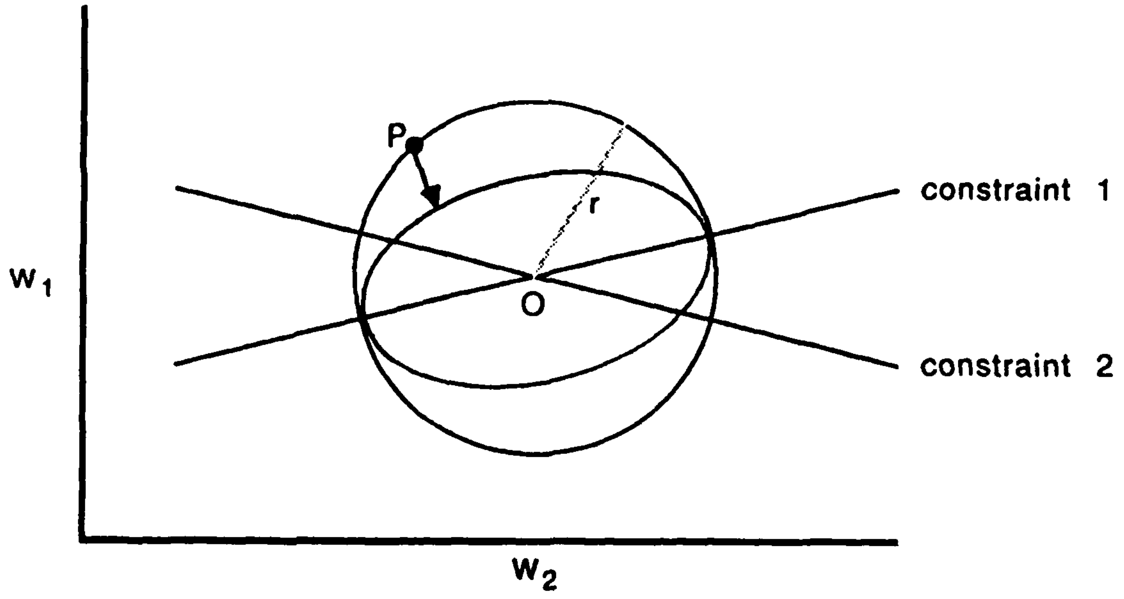


Figure 4: A plot of weight space for the simple network in figure 3.

Now, suppose we add to a network that has learned the associations a noise vector of length r that is selected at random from a distribution that is uniformly distributed over all possible orientations. The new combination of weights will be uniformly distributed over the circle shown in figure 4. If we now retrain an infinitessimal amount on association 1, we will move perpendicularly towards line 1. If we start from a point such as P that lies on one of the larger arcs of the circle, the movement towards line 1 will have a component *towards* line 2, whereas if we start from a point on one of the smaller arcs, the movement will have a component *away from* line 2. Thus, when we retrain a small amount on association 1, we are more likely than not to improve the performance on association 2, even though the associations are only randomly related to each other.

The expected positive transfer between retraining on one association and performance on the other can only be a result of starting the retraining from a point in weight space that has some special relationship to the solution point, O. We initially thought that the starting point for the retraining needed to be *near* the solution point, but the geometrical argument we have just given is independent of the magnitude, r, of the noise vector. The crucial property of the starting point is that it is selected at random from a distribution of points that are all the same distance from O, and selecting starting points in this way induces a bias towards starting points that cause positive transfer between the two associations that define the point O.

The positive transfer effect obviously disappears if the two associations are orthogonal, and so we might expect the effect to be rather small when the network is large, because randomly related high-dimensional vectors tend to be almost orthogonal. In fact, it is important to distinguish between two qualitatively different cases. If the activity levels of the input units have a mean of zero, most pairs of high-dimensional vectors will be approximately orthogonal and so the effect is small and the expected transfer from one retrained association to one unretrained one gets smaller as the dimensionality increases. If, however, the activity levels of the input units range between 0 and 1 (as in the simulation described above) random high-dimensional vectors are not close to orthogonal and we show in the next section that the expected transfer from one retrained association to one unretrained one is independent of the dimensionality of the vectors.

5. A mathematical analysis

It is hard to analyze the expected size of the transfer effect for networks with multiple layers of non-linear units or for tasks with complex regularities among the input-output associations. However, an analysis of the transfer effect in simple networks learning random associations may provide a useful guide to what can be expected in more complex cases. We therefore derive the magnitude of the expected transfer from one retrained association, A, to one unretrained association, B, in a network that has no hidden units and only one linear output unit.[1] We assume that the network has previously learned to produce the correct activity level in the output unit for two input vectors, **a** and **b**, and that it is now retraining on association A after independent gaussian noise has been added to each weight. The noise added to the i^{th} weight is g_i and is chosen from a distribution with mean 0 and standard deviation σ. We also assume that the retraining involves changing the weights by a fixed, infinitesimal amount in the direction of steepest descent in the error function $\frac{1}{2}(y_A - d_A)^2$ where y_A is the actual output of the output unit and d_A is the desired output that was achieved before the noise was added.

A good measure of the magnitude of the transfer effect is the the ratio of two improvements: the improvement in association B caused by retraining on A and the improvement in association B that would have been caused by retraining directly on B. If the retraining involves changing the weight vector by a fixed amount in the direction of steepest descent, this ratio is simply the cosine of the angle between the direction of steepest descent for association A and the direction of steepest descent for association B.[2]

If the original learning was perfect, all of the error in the output would be caused by the noise added

[1]For layered, feed-forward networks with no hidden units, each output unit has its own separate set of weights. So a network with many output units can be viewed as composed of many separate networks each of which has a single output unit.

[2]If the retraining involves multiplying the gradient by a coefficient, ε, to determine the weight change, the actual ratio may differ because the magnitudes of the gradients may differ for A and B. But this will not affect the *expected* ratio, so the analysis we give for the *expected* transfer is still valid.

to the weights, so

$$\frac{\partial E_A}{\partial y_A} = y_A - d_A = \sum_i a_i g_i$$

where a_i is the activity level of the i^{th} input unit in association A. So, for a weight, w_i, the derivatives of the error E_A for association A and E_B for association B are given by

$$\frac{\partial E_A}{\partial w_i} = \frac{\partial y_A}{\partial w_i} \cdot \frac{\partial E_A}{\partial y_A} \cdot a_i \cdot \sum_i a_i g_i , \qquad \frac{\partial E_B}{\partial w_i} = \frac{\partial y_B}{\partial w_i} \cdot \frac{\partial E_B}{\partial y_B} \cdot b_i \cdot \sum_i b_i g_i$$

Hence, the cosine of the angle, θ, between the directions of steepest descent for the two associations is given by

$$cos(\theta) = \frac{\sum_i a_i b_i \cdot \sum_i a_i g_i \cdot \sum_i b_i g_i}{\left[\sum_i a_i^2 \left(\sum_i a_i g_i\right)^2 \cdot \sum_i b_i^2 \left(\sum_i b_i g_i\right)^2\right]^{1/2}}$$

$$= \frac{\sum_i a_i b_i}{\left(\sum_i a_i^2 \cdot \sum_i b_i^2\right)^{1/2}} \cdot \frac{\sum_i a_i g_i \cdot \sum_i b_i g_i}{\left|\sum_i a_i g_i \cdot \sum_i b_i g_i\right|} \tag{1}$$

5.1. The zero-one case

The first part of equation 1 is simply the cosine of the angle between the two input vectors, and so it is independent of the weights. If the components of **a** and **b** are all 0 or 1 it can be written as

$$\frac{n_{\mathbf{ab}}}{(n_{\mathbf{a}} n_{\mathbf{b}})^{1/2}}$$

where $n_{\mathbf{a}}$ is the number of components that have value 1 in the vector **a**, and $n_{\mathbf{ab}}$ is the number that have value 1 in both **a** and **b**.

The second part of equation 1 depends on the weights. It always has a value of 1 or −1, depending on whether E_A and E_B have the same or opposite signs. Its numerator can be written as

$$\left(\sum_{i \in S_{\mathbf{ab}}} g_i + \sum_{i \in S_{\mathbf{a\bar{b}}}} g_i\right) \left(\sum_{i \in S_{\mathbf{ab}}} g_i + \sum_{i \in S_{\mathbf{\bar{a}b}}} g_i\right)$$

where $S_{\mathbf{ab}}$ is the set of input units that have value 1 in both **a** and **b**, $S_{\mathbf{a\bar{b}}}$ is the set that have value 1 in **a** and 0 in **b**, and $S_{\mathbf{\bar{a}b}}$ is the set that have value 0 in **a** and 1 in **b**. Since these sets are disjoint and the expected value of the products of independent zero-mean gaussians is 0, all the cross-products in the numerator of equation 1 vanish when we take expected values except for the term

$$\left\langle \sum_{i \in S_{\mathbf{ab}}} g_i \cdot \sum_{i \in S_{\mathbf{ab}}} g_i \right\rangle = \left\langle \sum_{i \in S_{\mathbf{ab}}} g_i^2 \right\rangle = n_{\mathbf{ab}} \sigma^2$$

So the expected value of equation 1 can be written as

$$\frac{n_{ab}\sigma^2}{\left\langle \left| \sum_i a_i g_i \cdot \sum_i b_i g_i \right| \right\rangle} = \frac{n_{ab}\sigma^2}{n_a^{1/2}\sigma\, n_b^{1/2}\sigma} = \frac{n_{ab}}{n_a^{1/2} n_b^{1/2}}$$

and so the expected value of $cos(\theta)$ is given by

$$\langle cos(\theta) \rangle = \frac{n_{ab}^2}{n_a n_b} \tag{2}$$

If each component of **a** and **b** has value 1 with probability 0.5 and value 0 otherwise, the expected value of $cos(\theta)$ is 0.25. So if we add random noise to the weights of a simple network that has learned 100 associations and then we retrain on 50 of them using very small weight changes, the ratio of the expected improvements on an unretrained and a retrained association at the start of the retraining will be

$$\frac{50 \times .25}{1 + 49 \times .25} = .943$$

This agrees well with simulations we have done using networks with no hidden units.

5.2. The 1, 0, -1 case

When the components of the input vectors can also take on values of -1, a modification of the derivation used above leads to

$$\langle cos(\theta) \rangle = \frac{(n_{agree} - n_{disagree})^2}{n_a n_b} \tag{3}$$

where n_{agree} is the number of input units for which $a_i b_i = 1$ and $n_{disagree}$ is the number of input units for which $a_i b_i = -1$.

For a pair of random vectors in which each component has a probability, p, of being a 1 and the same probability of being a -1, the expected value of the numerator of equation 3 is just the square of the expected length of a random walk in which each step has a probability of $2p^2$ of being to the left, $2p^2$ of being to the right, and $1-4p^2$ of being zero. The expected value of the numerator is therefore $4p^2 n$, where n is the dimensionality of the input vector. Since the denominator has an expected value of order n^2, the expected transfer effect is inversely proportional to n.

The decrease in the expected transfer between a single pair of vectors is balanced by the fact that larger networks can hold more associations. If the number of associations learned is proportional to the dimensionality of the input vectors, and if a constant *fraction* of the associations are retrained after adding noise to the weights, the ratio of the initial improvement on the unretrained associations to the improvement on the retrained associations is independent of the dimensionality.

5.3. How the transfer effect decreases during retraining

The analyses we have presented are for the transfer between random associations during the earliest stage of retraining. As retraining proceeds, the transfer effect diminishes. Figure 4 presents a simple geometrical explanation of why this occurs. At the start of retraining, the point in weight space lies somewhere on the circle, and the probability that the transfer will be positive is simply the fraction of the circumference that lies in the two larger arcs of the circle. Retraining on association 1 moves each point on the circle perpendicularly towards the line 1 by an amount proportional to its distance from 1, so after a given amount of retraining, each point on the circle will move to the corresponding point on the ellipse. The probability of positive transfer is now equal to the fraction of the circumference of the ellipse that lies in the two larger arcs.

6. Conclusions

This paper demonstrates and analyses a powerful and surprising transfer effect that was first described (but not analysed) by Hinton and Sejnowski (1986). The analysis applies just as well if there are no fast weights and the relearning takes place in the slow weights. The advantage of fast weights is that they allow this effect to be used for temporarily deblurring old memories without causing significant interference to new memories. When the fast weights decay away the newer memories are restored.

There are many anecdotal descriptions of phenomena that could be explained by the kind of mechanism we have proposed, but we know of no well controlled studies. We predict that if a two unrelated sets of associations are learned at the same time, and if the internal representations used for different associations share the same units, then retraining on one set of associations will substantially improve performance on the other set. One problem with this prediction is that with sufficient learning, connectionist networks tend to use different sets of units for representing unrelated items. A second problem is that even if the unretrained associations are enhanced, the effect may be masked by response competition due to facilitation of the responses to the retrained items. Nevertheless, the type of effect we have described can be very large in simulated networks and so a well designed experiment should be able to detect whether or not it occurs in people.

References

Feldman J.A. Dynamic connections in neural networks. *Biological Cybernetics*, 1982, *46*, 27-39.

Hartzell H.C. Mechanisms of slow synaptic potentials. *Nature*, 1981, *291*, 539-543.

Hinton G.E. and Sejnowski T.J. Learning and relearning in Boltzmann Machines. In D.E. Rumelhart, J.L. McClelland, and the PDP research group (Eds.) *Parallel distributed processing: Explorations in the microstructure of cognition. Volume I: Foundations*, Cambridge, MA: MIT Press, 1986.

Kupferman I. Modulatory actions of neurotransmitters. *Annual Review of Neuroscience*, 1979, *2*, 447-465.

Little W.A. and Shaw G.L. Statistical-theory of short and long-term memory. *Behavioral Biology*, 1975, *14(2)*, 115-133.

McClelland J.L. and Kawamoto A.H. Mechanisms of sentence processing: Assigning roles to constituents of sentences. In J.L. McClelland, D.E. Rumelhart, and the PDP research group (Eds.) *Parallel distributed processing: Explorations in the microstructure of cognition. Volume II: Psychological and biological models*, Cambridge, MA: MIT Press, 1986.

Rumelhart D.E., Hinton G.E. and Williams R.J. Learning internal representations by error propagation. In D.E. Rumelhart, J.L. McClelland, and the PDP research group (Eds.) *Parallel distributed processing: Explorations in the microstructure of cognition. Volume I: Foundations*, Cambridge, MA: MIT Press, 1986a.

Rumelhart D.E., Hinton G.E. and Williams R.J. Learning representations by back-propagating errors. *Nature*, 1986b, *323(9)*, 533-536.

Touretzky D.S. and Hinton G.E. Symbols among the neurons: Details of a connectionist inference architecture. *Proceedings, 9th International Joint Conference on Artificial Intelligence*, Los Angeles, August, 1985.

Von der Malsburg C. *The correlation theory of brain function*. Internal Report 81-2, Department of Neurobiology, Max-Plank-Institute for Biophysical Chemistry, P.O. Box 2841, Gottingen, F.R.G., 1981.